
Николай Прохоренок
Владимир Дронов
PRO
ПРОФЕССИОНАЛЬНОЕ ПРОГРАММИРОВАНИЕ
HTML
JavaScript
PHP и MySQL
Джентльменский набор Web-мастера
5-е издание
HTML 5
CSS 3
PHP 7.2
Web-сервер Apache
phpMyAdmin
AJAX
Примеры и советы из практики
Материалы на www.bhv.ru
bhv

Николай Прохоренок
Владимир Дронов

HTML JavaScript PHP и MySQL Джентльменский набор Web-мастера

5-е издание

Санкт-Петербург
«БХВ-Петербург»
2019

УДК 004.43+004.738.5
ББК 32.973.26-018.1
П84

Прохоренок, Н. А.

П84 HTML, JavaScript, PHP и MySQL. Джентльменский набор Web-мастера. — 5-е изд., перераб. и доп. / Н. А. Прохоренок, В. А. Дронов. — СПб.: БХВ-Петербург, 2019. — 912 с.: ил. — (Профессиональное программирование)

ISBN 978-5-9775-3986-9

Рассмотрены вопросы создания интерактивных Web-сайтов с помощью HTML, JavaScript, PHP и MySQL, форматирования Web-страниц при помощи CSS. Даны основы PHP и примеры написания типичных сценариев. Описаны приемы работы и администрирования баз данных MySQL при помощи PHP и программы phpMyAdmin. Особое внимание уделено созданию программной среды на компьютере разработчика и настройке Web-сервера Apache.

В 5-м издании содержится описание возможностей, предлагаемых PHP 7.2, новых инструментов JavaScript (включая рисование на холсте, средства геолокации и локальное хранилище данных) и всех нововведений, появившихся в актуальных на данный момент версиях HTML, CSS, Apache, MySQL и технологии AJAX.

Электронный архив содержит листинги примеров, а также руководство по созданию динамического сайта.

Для Web-разработчиков

УДК 004.43+004.738.5
ББК 32.973.26-018.1

Группа подготовки издания:

Руководитель проекта	*Евгений Рыбаков*
Зав. редакцией	*Екатерина Капалыгина*
Компьютерная верстка	*Ольги Сергиенко*
Дизайн обложки	*Марины Дамбиевой*

"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20.

ISBN 978-5-9775-3986-9

Оглавление

Введение

Если вы хотите научиться своими руками создавать сайты, свободно владеть HTML, CSS, JavaScript, PHP и MySQL, то эта книга для вас. Большинство подобных книг предлагают изучение или только клиентских технологий (HTML, CSS, JavaScript), или только серверных (PHP, MySQL). Но разделять эти технологии нельзя, т. к. они могут существовать только совместно, а значит, и изучать их нужно как единое целое.

Все главы книги расположены в порядке возрастания уровня сложности материала. Если вы начинающий Web-мастер, то книгу следует изучать последовательно, главу за главой. Если же материал какой-либо из глав был вами изучен ранее, то можно сразу переходить к следующей главе.

Что же можно создать с использованием описываемых технологий? Давайте кратко рассмотрим возможности этих технологий, а также содержание глав книги.

Язык разметки HTML, рассматриваемый в *главе 1*, позволяет задать местоположение элементов Web-страницы в окне Web-браузера. С помощью HTML можно отформатировать отдельные символы или целые фрагменты текста, вставить изображение, таблицу или форму, создать панель навигации с помощью карт-изображений, разделить окно Web-браузера на несколько областей, вставить гиперссылку и многое другое. А новая версия языка HTML — HTML 5 — даже позволяет поместить на страницу аудио- или видеоролик, который будет воспроизводиться самим Web-браузером, без необходимости устанавливать какие бы то ни было плагины.

При помощи каскадных таблиц стилей (CSS), о которых идет речь в *главе 2*, можно задавать точные характеристики практически всех элементов Web-страницы. Это позволяет контролировать внешний вид Web-страницы в окне Web-браузера и приближает возможности Web-дизайна к настольным издательским системам. Разработчик может указать параметры шрифта, цвет текста и фона, выравнивание, создать рамку и расположить элементы на странице произвольным образом. Новая версия CSS — CSS 3 — также предоставляет инструменты для задания градиентного фона, теней у текста и самого элемента страницы и даже для создания анимации.

У Web-страниц, созданных с использованием HTML и CSS, есть существенный недостаток — они являются статическими, т. е. не могут меняться, реагируя на действия пользователя. Внедрение в HTML-код программ, написанных на языке JavaScript, позволит «оживить» Web-страницу, сделать ее интерактивной или, дру-

гими словами, заставить взаимодействовать с пользователем. С помощью JavaScript можно обрабатывать данные формы до отправки на сервер, получать информацию о Web-браузере пользователя и его мониторе и соответствующим образом менять форматирование страницы, изменять любые элементы HTML-документа в ответ на какое-либо событие, создавать на Web-странице часы, показывающие текущее время с точностью до секунды, скрывать или отображать элементы Web-страницы и выполнять многие другие действия. Как все это сделать, рассказано в *главе 3*.

Глава 4 повествует о том, как установить и настроить специальное программное обеспечение для тестирования сайтов: Web-сервер Apache, среду для выполнения серверных скриптов, написанных на языке PHP, и сервер баз данных MySQL. С его помощью можно проверить работоспособность создаваемого сайта непосредственно на своем компьютере, еще до его публикации в Интернете.

Огромные возможности открывают серверные технологии, среди которых для целей этой книги выбран язык программирования PHP. Используя его, можно изменять получаемый Web-браузером HTML-код в зависимости от вводимых пользователем данных, типа и версии установленного Web-браузера и других факторов. Большое количество расширений и готовых программных продуктов, а также легкость освоения, сделали PHP очень популярным языком программирования для Интернета. С помощью PHP можно работать с файлами и каталогами, обрабатывать данные формы на сервере, рассылать письма, загружать файлы на сервер, создавать для каждого пользователя личный кабинет, программировать гостевые книги, форумы, блоги, интернет-магазины и многое другое. Писать программы на PHP мы научимся в *главе 5*.

На сегодняшний день ни один крупный портал не обходится без использования баз данных. В Web-разработках чаще всего применяется быстрая и обладающая большими возможностями система управления базами данных (СУБД) MySQL. С помощью MySQL можно эффективно добавлять, изменять и удалять данные, получать нужную информацию по запросу. Работа с MySQL, в том числе с базами данных этого формата из программ, написанных на PHP, обсуждается в *главе 6*.

В *главе 7* описывается технология AJAX, позволяющая программно подгружать с сервера произвольные данные без перезагрузки самой страницы. Применение технологии AJAX позволяет значительно расширить функциональность создаваемых сайтов.

В сопровождающем книгу электронном архиве, размещенном на сайте издательства «БХВ-Петербург» (см. *приложение*), собраны следующие материалы:

- описание процесса разработки полнофункционального Web-сайта с использованием всех изученных технологий. Это каталог сайтов, включающий личный кабинет для пользователей с защитой средствами PHP, а также личный кабинет для администратора, защищенный средствами сервера Apache;
- все листинги, встречающиеся в тексте книги. Настоятельно рекомендуем обязательно рассматривать все примеры из книги и вначале самостоятельно набирать код. При наборе вы создадите множество ошибок. Но именно умение находить эти ошибки сделает из вас настоящего Web-мастера.

Авторы желают вам приятного чтения и надеются, что эта книга станет верным спутником в вашей программистской практике.



ГЛАВА 1

Основы HTML 5. Создаем дизайн сайта

1.1. Первые шаги

HTML (*HyperText Markup Language*) — это язык разметки документа, описывающий форму отображения информации на экране компьютера. В настоящее время язык существует в нескольких версиях: HTML 4.01, XHTML и HTML 5.

Версию языка HTML 5 поддерживают сейчас следующие Web-браузеры:

- Microsoft Internet Explorer — начиная с версии 9;
- Mozilla Firefox — 4.0;
- Google Chrome — 6.0;
- Opera — 11.1;
- Apple Safari — 5.0.

Как можно видеть, HTML 5 поддерживается уже всеми современными Web-браузерами, поэтому мы будем изучать именно его, без оглядки на предыдущие версии.

Для новых тегов и параметров, появившихся в HTML 5, мы будем явно указывать версию языка. Поэтому, если у вас есть большое желание поддерживать старые версии Web-браузеров, то теги и параметры, обозначенные версией HTML 5, лучше не использовать или предусматривать стилизацию новых тегов — например, описывать их как блочные с помощью CSS-атрибута `display` (`display: block`) и/или создавать элементы в JavaScript.

> ***Примечание***
>
> Получить полную информацию о текущей поддержке тегов и параметров Web-браузерами можно на сайте **https://caniuse.com/**.

1.1.1. Основные понятия

При создании документа часто приходится выделять какую-либо часть текста полужирным шрифтом, изменять размер или цвет шрифта, выравнивать текст по центру страницы и т. д. В текстовом редакторе для этого достаточно выделить

нужный фрагмент и применить к нему форматирование. Например, чтобы пометить текст курсивом, нужно выделить его и нажать кнопку **Курсив**. На языке HTML тот же эффект достигается следующей строкой кода:

```
<i>Текст</i>
```

Комбинация символов `<i>` указывает, что текст надо выделить, начиная с этого места, а `</i>` отмечает конец выделенного фрагмента.

Комбинации символов `<i>` и `</i>` принято называть *тегами*. С помощью тегов описывается вся структура документа. Теги выделяются угловыми скобками < и >, между которыми указывается имя тега. Большинство тегов являются парными, т. е. состоят из открывающего тега (`<i>`) и соответствующего ему закрывающего (`</i>`). Закрывающий тег отличается наличием косой черты (/) перед его именем. Есть также теги, вообще не имеющие закрывающего тега, — например, тег переноса строки `<br>`.

Некоторые теги могут иметь *параметры* (иногда их называют *атрибутами*). Параметры указываются после имени тега через пробел в формате `параметр="значение"`. Если параметров несколько, то они приводятся через пробел:

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
```

В этом примере параметру `http-equiv` тега `<meta>` присвоено значение `Content-Type`, а параметру `content` — значение `text/html; charset=utf-8`.

Теги могут вкладываться друг в друга:

```
<p><i>Правильно</i></p>
```

При вложении тегов необходимо соблюдать последовательность их закрытия. Например, такой код использовать нельзя:

```
<p><i>Ошибка</p></i>
```

Теги бывают *блочными* и *строчными*. Блочный тег занимает всю ширину родительского элемента. Примером блочного тега является тег `<p>`, описывающий абзац. Строчный тег может быть расположен только внутри блочного тега. Примером строчного тега является тег `<i>`, указывающий, что фрагмент нужно выделить курсивом. Так как блочный тег `<p>` не может быть расположен внутри строчного тега `<i>`, следовательно, так вкладывать теги также нельзя:

```
<i><p>Ошибка</p></i>
```

Открывающие и закрывающие теги со всем их содержимым мы будем называть *элементами*. Например, в следующем примере описан элемент `p`:

```
<p>Текст абзаца</p>
```

> ***Примечание***
>
> В HTML названия тегов и параметров можно записывать в любом регистре, а в языке XHTML только в нижнем регистре. Рекомендуем всегда записывать теги в нижнем регистре. Именно так мы и будем записывать их в дальнейших примерах.

Просматривать HTML-документы можно с помощью специальных программ, которые называют *Web-браузерами*. Web-браузеры отображают документы с форматированием, выполненным на основе исходного кода, описывающего структуру документа.

Результат интерпретации HTML-документа, отображаемый в окне Web-браузера, называется *Web-страницей*. В отличие от HTML-документа Web-страница может содержать не только текст, но и графику, видео, звуковое сопровождение, может реагировать на действия пользователя и т. д. Кроме того, Web-страница может быть результатом интерпретации сразу нескольких HTML-документов.

Документы в формате HTML имеют расширение html или htm.

1.1.2. Редактор CKEditor

Прежде чем изучать язык HTML, советуем установить на компьютер редактор CKEditor. Этот редактор написан на языке программирования JavaScript и работает в Web-браузере.

Скачать CKEditor можно со страницы **https://ckeditor.com/ckeditor-4/download/**. Выберите полную версию (**Full Package**) и после закачки и распаковки архива запустите файл index.html (расположен в папке ckeditor\samples\) двойным щелчком левой кнопки мыши на значке файла. Можно также воспользоваться пунктом меню **Открыть** в Web-браузере.

На рис. 1.1 можно увидеть, как выглядит редактор CKEditor, запущенный в Web-браузере Firefox. Если вы раньше работали с текстовым редактором Microsoft Word, то большинство кнопок на панели инструментов будет вам знакомо. Принцип работы в CKEditor точно такой же, как и в Word. После ввода текста и его форматирования редактор автоматически генерирует HTML-код. Посмотреть исходный HTML-код можно, нажав кнопку **Источник** на панели инструментов (рис. 1.2).

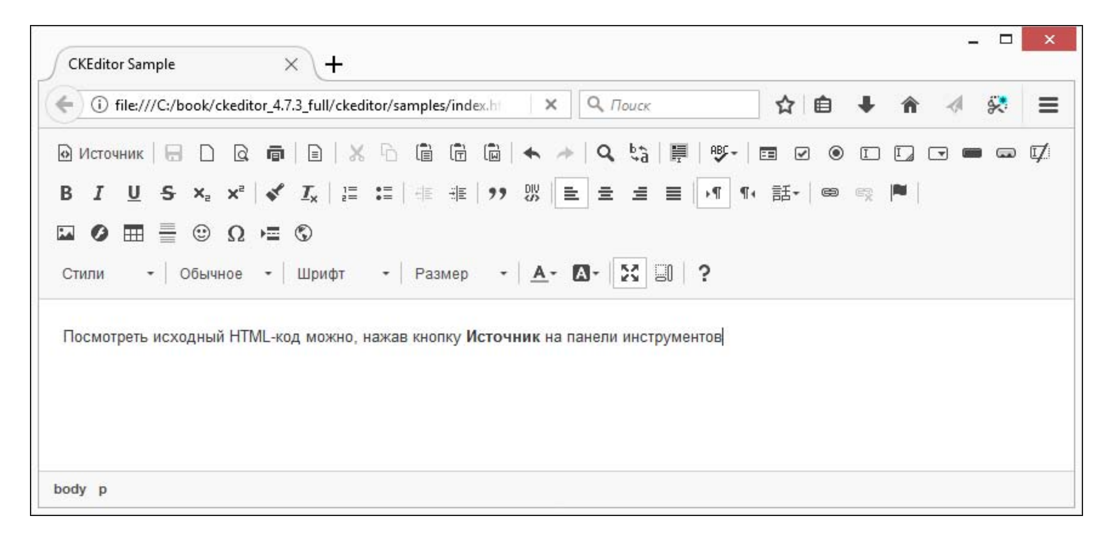


Рис. 1.1. Редактор CKEditor, запущенный в Web-браузере Firefox

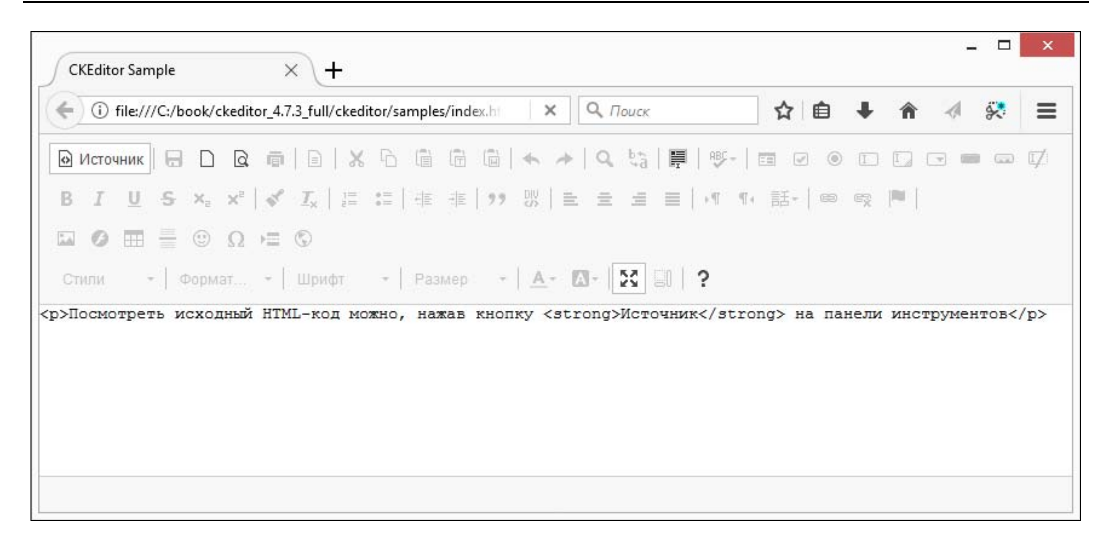


Рис. 1.2. Результат нажатия кнопки **Источник** в редакторе CKEditor

Следует заметить, что при изменении исходного HTML-кода автоматически изменяется и внешний вид документа.

1.1.3. Редактор Notepad++

Для создания HTML-документа можно воспользоваться любым текстовым редактором, например Блокнотом. Однако мы будем работать с кодировкой UTF-8, а Блокнот при сохранении в этой кодировке добавляет *метку порядка* байтов (от англ. Byte Order Mark, BOM). Эта метка может стать причиной ошибок при формировании заголовков ответа сервера из программы, написанной на языке PHP. Этот язык мы будем изучать далее, поэтому в качестве редактора кода на протяжении всего обучения воспользуемся программой Notepad++. Она позволяет корректно работать как с однобайтовой кодировкой windows-1251, так и с многобайтовой кодировкой UTF-8, а также имеет подсветку синтаксиса HTML, JavaScript, PHP и др.

Скачать программу Notepad++ можно абсолютно бесплатно со страницы **https://notepad-plus-plus.org/**. Из двух вариантов (архив и инсталлятор) советуем выбрать именно инсталлятор, т. к. при установке можно будет указать язык интерфейса программы. Установка Notepad++ предельно проста и в комментариях не нуждается.

Чтобы открыть какой-либо файл на редактирование, в меню **Файл** выбираем пункт **Открыть** или щелкаем правой кнопкой мыши на значке файла в Проводнике Windows и из контекстного меню выбираем пункт **Edit with Notepad++**.

> ***Примечание***
>
> Вместо Notepad++ можно воспользоваться редакторами PHP Expert Editor, Aptana Studio или NetBeans. Эти редакторы помимо подсветки синтаксиса предоставляют множество дополнительных функций. Тем не менее для быстрого редактирования файла удобнее пользоваться Notepad++.

1.1.4. Первый HTML-документ

Попробуем создать наш первый HTML-документ. Для его создания, как уже отмечалось, можно воспользоваться любым текстовым редактором. Мы же открываем редактор Notepad++ и создаем новый документ (меню **Файл | Новый**). В меню **Кодировки** устанавливаем флажок **Кодировать в UTF-8 (без BOM)**. Набираем код, представленный в листинге 1.1, а затем в меню **Файл** выбираем пункт **Сохранить как**. В открывшемся окне выбираем папку, например, C:\book, в строке **Имя файла** вводим `test.html`, а из списка **Тип файла** выбираем пункт **All types (*.*)**. Нажимаем кнопку **Сохранить**. После сохранения файла в заголовке окна редактора должен появиться путь: C:\book\test.html. Если окажется, что после фрагмента html стоит точка и какое-либо другое расширение, например txt, то при сохранении была допущена ошибка. Такая ошибка очень часто возникает при сохранении файла в редакторе Блокнот.

Листинг 1.1. Первый HTML-документ

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Заголовок страницы</title>
</head>
<body>
<p>Привет, мир!</p>
</body>
</html>
```

Чтобы посмотреть результат, нужно открыть HTML-документ в Web-браузере. В этом издании книги мы будем пользоваться Web-браузером Mozilla Firefox (на момент подготовки издания текущей была версия 56). Можно пользоваться и другим Web-браузером, но процесс работы с другим Web-браузером будет отличаться, и вы станете путаться, не зная, что делать. Поэтому на время обучения установите Web-браузер Mozilla Firefox последней версии, а уже после обучения можете вернуться к своему любимому Web-браузеру.

Итак, запускаем Web-браузер Mozilla Firefox и переходим в главное меню. Выбираем пункт **Открыть файл**, в открывшемся окне находим наш файл test.html и нажимаем кнопку **Открыть**. Если все сделано правильно, то в окне Web-браузера вы увидите надпись «Привет, мир!», а в строке заголовка — надпись «Заголовок страницы». Теги в окне Web-браузера не отображаются!

> ***Примечание***
>
> Если в главном меню браузера вы не найдете пункт **Открыть файл**, то выберите пункт **Изменить**, найдите в левой части открывшегося окна пункт **Открыть файл** и перетащите его с помощью мыши в область меню. Точно таким же способом добавьте в меню пункт **Кодировка текста**. После чего нажмите кнопку **Выход из настройки** для применения изменений.

Если Web-браузер Firefox является Web-браузером по умолчанию, то открыть HTML-документ можно с помощью двойного щелчка левой кнопкой мыши на значке файла. Если это не так, то щелкаем правой кнопкой мыши на значке файла и из контекстного меню выбираем пункт **Открыть с помощью | Выбрать программу**. В открывшемся окне находим Web-браузер Mozilla Firefox и устанавливаем флажок **Использовать это приложение для всех файлов html**. Либо в настройках Web-браузера делаем его установленным по умолчанию.

Можно также в адресной строке Web-браузера набрать команду `file:///C:/book/test.html` и нажать клавишу <Enter>.

Если вместо русского текста в окне Web-браузера отобразились непонятные символы, то следует вначале проверить кодировку файла с HTML-документом. Для этого в редакторе Notepad++ отображаем содержимое файла и открываем меню **Кодировки**. Флажок должен быть установлен у пункта **Кодировать в UTF-8 (без BOM)**. Если это не так, то выбираем пункт **Преобразовать в UTF-8 (без BOM)**, сохраняем файл и пытаемся открыть его снова.

Если русские буквы все равно отображаются неправильно, то проверяем кодировку в следующей строке:

```
<meta charset="utf-8">
```

Для совместимости со старыми Web-браузерами эту строку можно записать так:

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
```

Если опять не помогло, то смотрим какую кодировку выбрал Web-браузер. Для этого открываем главное меню и выбираем пункт **Кодировка текста**. Флажок должен стоять у пункта **Юникод**.

Теперь попробуем изменить заголовок в окне Web-браузера. Открываем файл с HTML-документом в программе Notepad++ и изменяем строчку:

```
<title>Заголовок страницы</title>
```

на:

```
<title>Моя первая Web-страница</title>
```

Сохраняем файл (меню **Файл | Сохранить**), возвращаемся в Web-браузер и обновляем Web-страницу. Обновить страницу в Firefox можно следующими способами:

- нажатием кнопки **Обновить текущую страницу** в адресной строке;
- щелчком правой кнопкой мыши на заголовке текущей вкладки и выбором из контекстного меню пункта **Обновить вкладку**;
- нажатием на клавиатуре комбинации клавиш <Ctrl>+<R> или клавиши <F5>.

В результате надпись в строке заголовка заменится на «Моя первая Web-страница».

Таким образом, изменяя что-либо в исходном коде, можно визуально оценивать результаты произведенных действий. Алгоритм такой: открываем исходный код, вносим корректировку, сохраняем, а затем обновляем Web-страницу.

1.1.5. Просмотр исходного HTML-кода в Web-браузере

Чтобы посмотреть исходный HTML-код загруженной Web-страницы, следует щелкнуть правой кнопкой мыши в любом месте окна Web-браузера и выбрать из контекстного меню пункт **Исходный код страницы**. В некоторых случаях результат этого действия может быть разным. Так, если Web-страница состоит из нескольких HTML-документов, то от места щелчка зависит, код какого HTML-документа будет отображен.

Чтобы посмотреть код только какого-либо фрагмента, достаточно выделить его в окне, щелкнуть на выделении правой кнопкой мыши и из контекстного меню выбрать пункт **Исходный код выделенного фрагмента**.

Исходный HTML-код будет отображен на отдельной вкладке, причем его синтаксис подсвечивается. Если какой-либо фрагмент подсвечен красным цветом, то в нем содержится ошибка, которую нужно исправить. К ошибкам внутри HTML-кода Web-браузеры часто относятся снисходительно и исправляют их самостоятельно, но часто не так, как вам бы этого хотелось. Поэтому обращайте внимание на фрагменты, подсвеченные красным цветом — это подсказка для вас.

Web-браузеры запускают код в так называемой «песочнице». Это означает, что код внутри Web-страницы не может напрямую взаимодействовать с файловой системой компьютера. Сделано это из соображений безопасности. Поэтому, если HTML-документ опубликован в Интернете, то можно лишь созерцать его исходный код, а вот изменить его нельзя. Можно лишь скопировать его и вставить куда-либо или сохранить в файл, выбрав из контекстного меню пункт **Сохранить как**. Последнее действие выполняет лично пользователь, а не код внутри Web-страницы, поэтому ограничения «песочницы» здесь не действуют.

1.1.6. Инструменты разработчика

Web-браузер Mozilla Firefox содержит **Инструменты разработчика**, которые позволяют не только просматривать исходный код, но и изменять значения различных параметров тегов, атрибутов стиля и т. д. При этом все изменения сразу применяются в окне Web-браузера.

Чтобы открыть панель **Инструменты разработчика**, нужно в главном меню выбрать пункт **Разработка | Инструменты разработчика** или нажать комбинацию клавиш <Shift>+<Ctrl>+<I>. На этой панели сейчас нас интересует вкладка **Инспектор**. Поэтому, чтобы отобразить ее сразу, в главном меню выбираем пункт **Разработка | Инспектор** или нажимаем комбинацию клавиш <Shift>+<Ctrl>+<C>.

По умолчанию панель отобразится в нижней части окна. Это поведение можно изменить с помощью значков в правой части заголовка панели. Например, можно отобразить панель в отдельном окне или прикрепить ее к боковой части окна. В заголовке окна содержится также значок с изображением шестеренки (пункт **Настройки инструментов**). Щелчок на нем откроет панель с различными настройками.

На вкладке **Инспектор** отображается структура HTML-документа в виде дерева (рис. 1.3), а на вкладке **Разметка** можно увидеть блочную структуру каждого элемента (если вкладка не отображается, то нужно щелкнуть левой кнопкой мыши на значке **Развернуть панель**). При наведении указателя мыши на какой-либо элемент он подсвечивается в окне Web-браузера, что позволяет определить его местоположение визуально. Чтобы сразу перейти к элементу внутри HTML-кода, щелкаем на значке **Выбрать элемент из страницы** в заголовке панели, а затем наводим указатель мыши на элемент в окне Web-браузера и щелкаем на нем левой кнопкой мыши, — строка HTML-кода, соответствующая этому элементу, на панели **Инспектор** будет выделена.

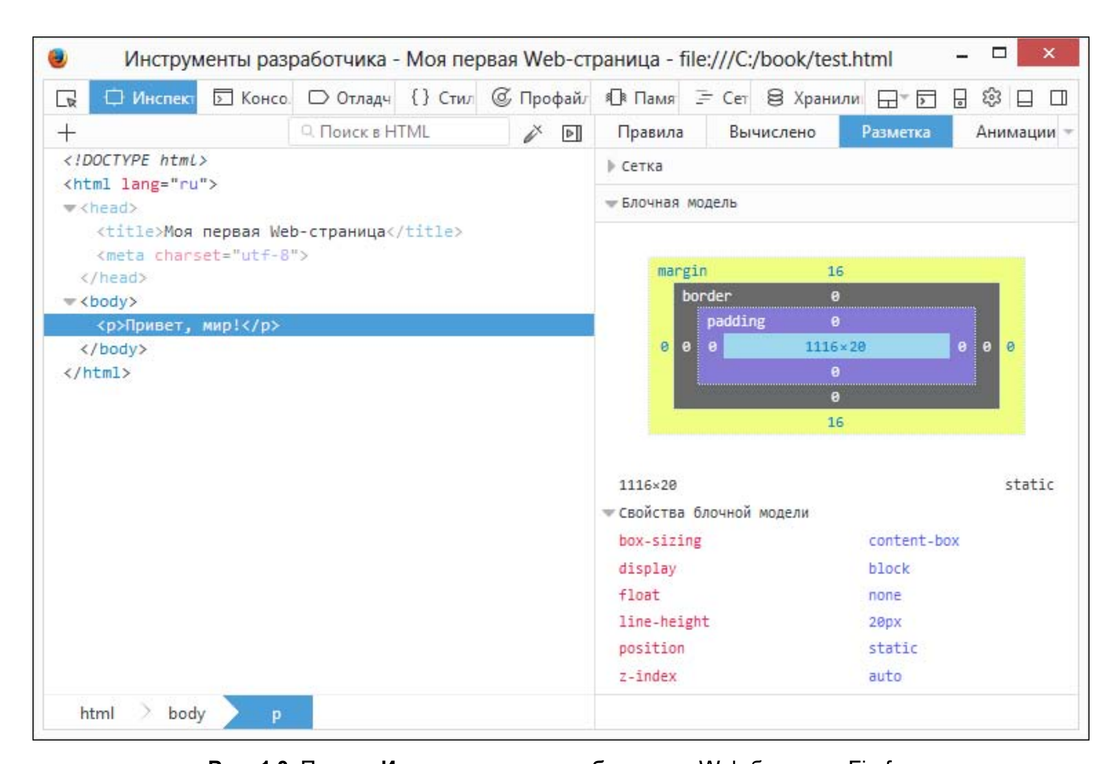


Рис. 1.3. Панель **Инструменты разработчика** в Web-браузере Firefox

Все значения на этой панели можно менять и сразу видеть результат в окне. Например, изменим текст абзаца. Для этого найдем элемент `p` и сделаем двойной щелчок левой кнопкой мыши на тексте — текст станет доступен для редактирования. Вводим любой другой текст, нажимаем клавишу <Enter> и любуемся проделанной работой в окне Web-браузера.

Если нужно изменить не только текст элемента, но и поменять HTML-код, то щелкаем правой кнопкой мыши на элементе и из контекстного меню выбираем пункт **Править как HTML**. После этого станет доступным поле, в котором можно произвести изменения. Например, изменим код абзаца следующим образом:

```
<p><i>Текст выделен курсивом</i></p>
```

Точно таким же образом можно отобразить в виде текста весь HTML-код. Для этого достаточно щелкнуть правой кнопкой мыши на открывающем теге `<html>` и из контекстного меню выбрать пункт **Править как HTML**. Чтобы скопировать весь код, вначале выделяем его, а затем нажимаем комбинацию клавиш <Ctrl>+<C>, — код будет скопирован в буфер обмена, и его можно будет вставить в любое другое место — например, сохранить в файл. Аналогичное действие позволяет выполнить пункт из контекстного меню **Копировать | Внешний HTML**.

Пользоваться панелью **Инструменты разработчика** мы будем очень часто, особенно при изучении языка программирования JavaScript, поэтому способы отображения этой панели нужно знать наизусть.

1.1.7. Комментарии в HTML-коде

Текст, заключенный между тегами `<!--` и `-->`, Web-браузером не отображается. Заметим, что это нестандартная пара тегов, т. к. открывающий тег не имеет закрывающей угловой скобки, а в закрывающем теге отсутствует открывающая угловая скобка:

```
<!-- Текст комментария -->
```

> ***Совет***
> Использование комментариев в исходном коде позволит быстро найти нужный фрагмент. Это особенно важно для начинающих Web-разработчиков.

1.2. Структура HTML-документа

Итак, мы изучили технологию создания HTML-документов, научились сохранять, отображать и изменять исходный код. Пришла пора вернуться к языку HTML. В листинге 1.2 представлена структура, характерная для любого HTML-документа.

Листинг 1.2. Структура HTML-документа

```
<!DOCTYPE> <!-- Объявление формата документа -->
<html>
   <head>
      <!-- Техническая информация о документе -->
   </head>
   <body>
      <!-- Основная часть документа -->
   </body>
</html>
```

1.2.1. Тег *<!DOCTYPE>*. Объявление формата документа

Тег `<!DOCTYPE>` позволяет Web-браузеру определить формат файла и правильно отобразить все его инструкции. Для HTML 5 инструкция выглядит очень просто:

```
<!DOCTYPE html>
```

Допустимые форматы для HTML 4.01:

- `Strict` — строгий формат. Не содержит тегов и параметров, помеченных как устаревшие или не одобряемые. Объявление формата:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
         "http://www.w3.org/TR/html4/strict.dtd">
```

- `Transitional` — переходный формат. Содержит устаревшие теги в целях совместимости и упрощения перехода со старых версий HTML. Объявление формата:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
                       "http://www.w3.org/TR/html4/loose.dtd">
```

- `Frameset` — аналогичен переходному формату, но содержит также теги для создания фреймов. Объявление формата:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
                "http://www.w3.org/TR/html4/frameset.dtd">
```

Если тег `<!DOCTYPE>` не указан, то некоторые Web-браузеры переходят в режим совместимости. В этом режиме может отличаться тип блочной модели. Поэтому при отсутствии тега `<!DOCTYPE>` разные Web-браузеры могут по-разному отображать Web-страницу.

1.2.2. Тег *<html>*

Весь текст HTML-документа расположен между тегами `<html>` и `</html>`. Тег `<html>` может иметь параметры. В большинстве случаев эти параметры универсальные, и мы их рассмотрим позже. Сейчас же будем указывать только параметр `lang`, задающий код языка Web-страницы:

```
<html lang="ru">
```

Значение `ru` параметра `lang` означает русский язык.

HTML-документ состоит из двух разделов: *заголовка* (между тегами `<head>` и `</head>`) и *содержательной части* (между тегами `<body>` и `</body>`).

1.2.3. Раздел *HEAD*. Техническая информация о документе

Раздел `HEAD` содержит техническую информацию о странице: заголовок, ее описание и ключевые слова для поисковых машин, данные об авторе и времени создания страницы, базовом адресе страницы, кодировке и т. д.

Единственным обязательным тегом в разделе HEAD является тег <title>. Текст, расположенный между тегами <title> и </title>, отображается в строке заголовка Web-браузера или в строке заголовка вкладки. Длина заголовка не должна превышать 60 символов, иначе он полностью не поместится:

```
<title>Заголовок страницы</title>
```

> ***Совет***
>
> Очень часто текст между тегами <title> и </title> используется в результатах, выдаваемых поисковыми порталами, в качестве текста ссылки на эту страницу. Соответственно, заголовок должен максимально полно описывать содержание страницы. Не следует писать что-то вроде «Главная страница», «Первая страница» и т. п.

С помощью одинарного тега <meta> можно задать описание содержимого страницы и ключевые слова для поисковых машин. Если текст между тегами <title> и </title> используется в качестве текста ссылки на эту страницу, то описание из тега <meta> может быть отображено под ссылкой:

```
<meta name="description" content="Описание содержимого страницы">
<meta name="keywords" content="Ключевые слова через запятую">
```

Также с помощью тега <meta> можно указать кодировку текста:

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
```

В HTML 5 указать кодировку можно гораздо проще:

```
<meta charset="utf-8">
```

Для автоматической перезагрузки страницы через заданный промежуток времени следует воспользоваться свойством refresh тега <meta>:

```
<meta http-equiv="refresh" content="30">
```

В этом примере страница будет перезагружена через 30 секунд. Если необходимо сразу перебросить посетителя на другую страницу, то можно указать ее интернет-адрес (URL) в параметре url:

```
<meta http-equiv="refresh" content="0; url=http://mail.ru/">
```

> ***Примечание***
>
> В разделе HEAD могут быть расположены также теги <base>, <link>, <script>, <style> и некоторые другие. Эти теги мы рассмотрим по мере изучения материала.

1.2.4. Файл *favicon.ico*

Когда посетители добавляют сайт в **Избранное**, Web-браузеры запрашивают с сервера файл favicon.ico. Этот файл должен содержать логотип сайта в виде значка 16×16 или 32×32 пиксела. В одном файле может находиться несколько форматов значка сразу. Если файл не найден, то в журнал регистрации ошибок сервера записывается сообщение об ошибке 404 (файл не найден), а в строке в **Избранном** отображается стандартный значок Web-браузера. Если значок найден, то он отобразится в **Избранном** (рис. 1.4) и в заголовке вкладки перед названием Web-страницы.

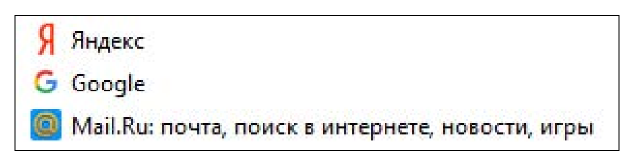


Рис. 1.4. Пользовательские значки в **Избранном**

Некоторые поисковые порталы (например, Яндекс) отображают значок в результатах поиска.

Файл favicon.ico по умолчанию должен находиться в корневой папке сайта и быть доступным по адресу **http://<Имя сайта>/favicon.ico**. В раздел `HEAD` HTML-документа можно добавить следующие строчки, задающие точное местоположение значка:

```
<link rel="icon" href="/favicon.ico" type="image/x-icon">
<link rel="shortcut icon" href="/favicon.ico" type="image/x-icon">
```

Файлы favicon.ico создаются в специализированных редакторах. Дополнительную информацию о файле favicon.ico и методах его создания можно получить на сайтах **http://favicon.ru/** и **http://favicon.com/**.

1.2.5. Файл *robots.txt*

В некоторых случаях следует запретить индексацию отдельных страниц сайта или всего сайта — например, если он предназначен только для членов семьи. Следует также учитывать, что поисковые роботы создают большой трафик, т. к. запрашивают все страницы сайта без передышки. Это обстоятельство может значительно нагрузить сервер.

Для управления индексацией конкретной страницы можно воспользоваться тегом `<meta>`:

```
<meta name="robots" content="<Индексация>, <Переход по ссылкам>">
```

В параметре `content` указывается комбинация следующих значений:

- `index` — индексация разрешена;
- `noindex` — индексация запрещена;
- `follow` — разрешено переходить по ссылкам, которые находятся на этой Web-странице;
- `nofollow` — запрещено переходить по ссылкам;
- `all` — комбинация `index` плюс `follow`;
- `none` — комбинация `noindex` плюс `nofollow`.

Приведем ряд примеров:

- индексация и переход по ссылкам разрешены:

  ```
  <meta name="robots" content="index, follow">
  ```

- индексация разрешена, а переход по ссылкам запрещен:

  ```
  <meta name="robots" content="index, nofollow">
  ```

- индексация и переход по ссылкам запрещены:

  ```
  <meta name="robots" content="noindex, nofollow">
  ```

Но этот метод не снизит нагрузку на сервер. Чтобы этого добиться, необходимо в корневой папке сайта создать файл robots.txt (файл должен быть доступен по адресу **http://<Имя сайта>/robots.txt**). Содержимое файла выглядит следующим образом:

```
User-agent: *
Disallow: /file.html
Disallow: /user/
```

В директиве `User-agent` можно задать конкретное название робота или указать символ `*`, который означает, что приведенная информация относится ко всем роботам. Узнать название робота можно по полю `user-agent` в логах сервера.

Каждая строка `Disallow` определяет файл или каталог, который запрещен для индексации указанным роботом.

Перед индексацией сайта все роботы обязаны ознакомиться с содержимым этого файла. Даже если все страницы сайта разрешены для индексации, все равно следует создать пустой файл robots.txt и поместить его в корневой папке сайта, т. к. отсутствие этого файла приведет к ошибке 404 (файл не найден). Отсутствие файла означает, что индексация разрешена.

Чтобы запретить индексацию всего сайта, необходимо разместить файл со следующими директивами:

```
User-agent: *
Disallow: /
```

1.2.6. Раздел *BODY*. Основная часть документа

В этом разделе располагается все содержимое документа. Большинство тегов, рассмотренных в этой главе книги, должны находиться именно между тегами `<body>` и `</body>`.

Следует отметить, что в HTML 4.01 в формате `Strict` содержимое тега `<body>` должно быть расположено внутри блочных элементов, например: `<p>`, `<div>` или др.:

```
<p>Текст документа</p>
<div>Текст документа</div>
```

В HTML 5 этого не требуется, даже теги `<html>` и `<body>` являются необязательными. Например, следующий HTML-документ является валидным (однако так лучше не делать!):

```
<!DOCTYPE html>
<head>
   <title>Заголовок страницы</title>
```

```
    <meta charset="utf-8">
</head>
Привет, мир!
```

В HTML 4.01 в формате `Transitional` тег `<body>` имеет параметры `bgcolor` (задает цвет фона Web-страницы), `background` (позволяет задать фоновый рисунок для документа), `alink` (определяет цвет активной ссылки), `link` (устанавливает цвет еще не просмотренных ссылок), `vlink` (определяет цвет уже просмотренных ссылок), `text` (устанавливает цвет текста) и др. В HTML 5 все эти параметры отсутствуют. Вместо них следует пользоваться стилями CSS (листинг 1.3).

Листинг 1.3. Тег <body>

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Заголовок страницы</title>
   <style>
      body {
         /* Цвет фона Web-страницы */
         background-color: green;
         /* Фоновый рисунок */
         background-image: url(photo.jpg);
         /* Цвет текста */
         color: red
      }
      /* Цвет еще не просмотренных ссылок */
      a:link { color: #0000EE }
      /* Цвет уже просмотренных ссылок */
      a:visited { color: #008000 }
      /* Цвет активной ссылки */
      a:active { color: #FF0000 }
   </style>
</head>
<body>
<p>Текст документа</p>
<p><a href="https://www.google.ru/">Текст ссылки</a></p>
</body>
</html>
```

В этом примере мы воспользовались парным тегом `<style>` и внутри него добавили различные стили как для тега `<body>`, так и для тега `<a>`. Более подробно CSS стили мы рассмотрим в следующей главе.

У тега `<body>` существуют и другие параметры, которые мы будем рассматривать по мере изучения языка.

1.3. Разделение Web-страницы на фрагменты

Элементы Web-страницы бывают *блочными* и *строчными*. Блочный элемент занимает всю ширину родительского элемента. Строчный элемент может быть расположен только внутри блочного элемента. С помощью блочных элементов мы можем разделить Web-страницу на логические фрагменты — например, выполнить форматирование текста с помощью заголовков и абзацев.

1.3.1. Заголовки

Заголовки могут иметь шесть различных размеров:

```
<hx>Заголовок</hx>
```

где `x` — число от 1 до 6.

Заголовок с номером 1 является самым крупным:

```
<h1>Самый крупный заголовок</h1>
```

Заголовок с номером 6 является самым мелким:

```
<h6>Самый мелкий заголовок</h6>
```

В HTML 4.01 в формате `Transitional` теги `<hx>` имеют параметр `align`, с помощью которого можно управлять горизонтальным выравниванием текста внутри заголовка. В HTML 5 этого параметра нет, вместо него нужно использовать атрибут стиля `text-align`:

```
<h1 style="text-align: center">Текст по центру</h1>
```

В этом примере мы воспользовались параметром `style`, который имеют почти все теги.

1.3.2. Разделение на абзацы

Тег `<p>` позволяет разделить текст на отдельные абзацы. При этом перед абзацем и после него добавляется пустое пространство:

```
<h1>Заголовок</h1>
<p>Абзац с выравниванием по левому краю</p>
<p style="text-align: center">Абзац с выравниванием по центру</p>
<p style="text-align: justify">Абзац с выравниванием по ширине</p>
```

Закрывающий тег `</p>` не обязателен, поэтому и такой код является валидным:

```
<h1>Заголовок</h1>
<p>Абзац 1
<p>Абзац 2
<p>Абзац 3
```

В этом случае концом абзаца считается следующий открывающий тег `<p>` или любой другой блочный элемент.

Парный тег <blockquote> создает абзац с длинной цитатой. Его блок будет иметь меньшую ширину, чем обычный абзац, и выравнивание по центру.

```
<blockquote>Длинная цитата</blockquote>
```

Тег <blockquote> может содержать параметр cite, задающий ссылку на страницу с оригинальным текстом.

1.3.3. Тег <*div*>

Тег <div> является универсальным блочным элементом. С его помощью можно сгруппировать несколько элементов страницы в один и применить к группе различные стили. Кроме того, тег <div> используется для блочной верстки Web-страницы (листинг 1.4).

Листинг 1.4. Тег <div>

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Элемент DIV</title>
</head>
<body>
   <div id="header">
      <h1>&quot;Шапка&quot; страницы</h1>
   </div>
   <div id="nav">
      <p><a href="page2.html">Страница 2</a></p>
      <p><a href="page3.html">Страница 3</a></p>
   </div>
   <div id="main">
      <p>Основное содержимое страницы</p>
   </div>
   <div id="footer">
      <p>&quot;Подвал&quot; страницы</p>
   </div>
</body>
</html>
```

1.3.4. Семантическая разметка в HTML 5

Итак, в листинге 1.4 мы создали блочную верстку с помощью тегов <div>. Каждый тег содержит параметр id, задающий уникальный идентификатор элемента. Этот параметр имеют практически все теги. Благодаря этому параметру мы можем обратиться к элементу из программы на JavaScript или применить к элементу стили.

Давайте посмотрим на первый элемент:

```
<div id="header">
  <h1>&quot;Шапка&quot; страницы</h1>
</div>
```

По значению параметра id человек догадается о предназначении элемента, а вот поисковый робот видит только безликий элемент div и не догадывается о его предназначении. Чтобы это исправить, в HTML 5 были добавлены средства *семантической разметки страниц*. Под семантической разметкой подразумевается разделение содержимого страницы на части, каждая из которых имеет особое значение: заголовок страницы, панель навигации, основное содержимое, часть основного содержимого, «подвал» и др.

Перепишем предыдущий пример:

```
<header>
   <h1>&quot;Шапка&quot; страницы</h1>
</header>
```

Мы заменили безликий элемент div блочным элементом header, имеющим определенный смысл. Теперь поисковый робот знает, с чем имеет дело, и сможет правильно обработать содержимое элемента.

Для семантической разметки в HTML 5 предусмотрены следующие блочные теги:

- <header> — «шапка» статьи или самой страницы;
- <nav> — набор гиперссылок для навигации по содержимому статьи или самому сайту (панель навигации);
- <main> — блок основного содержимого страницы;
- <footer> — «подвал» статьи или самой страницы;
- <section> — значимая часть материала (например, раздел документа). Раздел должен содержать заголовок;
- <article> — независимый и самодостаточный фрагмент основного содержимого страницы: отдельная статья, запись форума, блога или комментарий;
- <aside> — примечание к статье, обычно располагающееся сбоку от основного текста;
- <figure> — иллюстрация к статье: обычное графическое изображение, аудио-, видеоролик или даже фрагмент текста;
- <figcaption> — подпись к иллюстрации. Может присутствовать только в теге <figure>.

С помощью строчного тега <mark> можно пометить важный фрагмент текста. Web-браузер Firefox устанавливает для такого фрагмента желтый цвет фона (как бы выделяет фрагмент текста маркером).

Пример использования семантической разметки приведен в листинге 1.5.

Листинг 1.5. Семантическая разметка

```
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Семантическая разметка</title>
</head>
<body>
   <header>
      <h1>&quot;Шапка&quot; страницы</h1>
   </header>
   <nav>
      <p><a href="page2.html">Страница 2</a></p>
      <p><a href="page3.html">Страница 3</a></p>
   </nav>
   <section>
      <h2>Заголовок раздела</h2>
      <p>Основное содержимое</p>
      <article>
         <h3>Заголовок статьи</h3>
         <p>Текст статьи.
            <mark>Важный фрагмент текста</mark>
         </p>
         <figure>
            <p><img src="photo.jpg" alt=""></p>
            <figcaption>Иллюстрация</figcaption>
            </figure>
         <aside>
            <p>Примечание к статье</p>
         </aside>
      </article>
   </section>
   <footer>
      <p>&quot;Подвал&quot; страницы</p>
   </footer>
</body>
</html>
```

Web-браузеры сейчас никак не выделяют визуально блочные теги семантической разметки. Однако мы всегда сможем привязать к ним стили, чтобы задать нужное нам оформление (о стилях будет рассказано в *главе 2*).

Чтобы старые версии Web-браузеров правильно отображали новые элементы, нужно объявить их блочными с помощью атрибута стиля `display`:

```
<style type="text/css">
main { display: block }
</style>
```

Для браузера Internet Explorer следует дополнительно создать элемент:

```
<script type="text/javascript">
document.createElement("main");
</script>
```

1.3.5. Тег *<details>*

Тег <details> в HTML 5 позволяет скрыть или отобразить какой-либо фрагмент страницы. По умолчанию содержимое элемента скрыто. Чтобы отобразить содержимое, нужно щелкнуть левой кнопкой мыши на заголовке, реализуемом с помощью тега <summary>, или добавить параметр open. Чтобы опять скрыть содержимое, нужно повторно щелкнуть мышью на заголовке:

```
<details open>
   <summary>Развернуть или свернуть</summary>
   <p>Скрытый текст</p>
</details>
```

Оба этих тега поддерживаются браузерами Chrome версии 12+, Opera версии 15+, Safari версии 6+ и Firefox версии 49+. Браузеры Internet Explorer и Microsoft Edge тег <details> не поддерживают.

1.3.6. Горизонтальная линия

Одинарный тег <hr> позволяет провести горизонтальную линию. По умолчанию линия занимает почти всю ширину родительского элемента и выравнивается по центру.

В HTML 4.01 в формате Transitional тег <hr> имеет параметры size (толщина линии), width (длина линии), align (выравнивание линии) и noshade (присутствие этого параметра отменяет рельефность линии). В HTML 5 этих параметров нет, вместо них нужно использовать стили:

```
<p>Со значениями по умолчанию</p>
<hr>
<p>Высота 5 px, ширина 90 процентов, с рамкой</p>
<hr style="height: 5px; width: 90%">
<p>Ширина 200 px, с выравниванием по правой стороне</p>
<hr style="width: 200px; margin: 0 0 0 auto">
<p>Высота 5 px, ширина 200 px, без рамки,
   с выравниванием по центру, красный цвет фона</p>
<hr style="height: 5px; width: 200px; border: 0; background: red">
```

1.4. Форматирование текста

Как уже говорилось, HTML — это *язык разметки*. Следовательно, важно уметь форматировать отдельные символы, а также целые фрагменты текста. Форматирование текста внутри блока выполняется строчными тегами.

1.4.1. HTML-эквиваленты

Прежде чем изучать теги, рассмотрим возможность отображения специальных символов. Такими символами, например, являются знаки меньше (<) и больше (>), т. к. с помощью этих символов описываются HTML-теги. Для отображения специальных символов используются так называемые *HTML-эквиваленты*. Например, для вывода такого текста:

```
Текст между тегами <title> и </title> используется в результатах,
выдаваемых поисковым порталом.
```

необходимо написать так:

```
Текст между тегами &lt;title&gt; и &lt;/title&gt; используется
в результатах, выдаваемых поисковым порталом.
```

В этом примере мы заменили знак меньше (<) на <, а знак больше (>) — на >. Приведем наиболее часто используемые HTML-эквиваленты:

- < — знак меньше (<);
- > — знак больше (>);
- & — амперсанд (&);
-   — неразрывный пробел;
- " — кавычка (");
- © — знак авторских прав (©);
- ® — знак зарегистрированной торговой марки (®);
- ™ — торговая марка (™).

1.4.2. Перевод строки

Для разделения строк используется одинарный тег
.

Если в HTML-документе набрать текст:

```
<p>Строка1
Строка2
Строка3</p>
```

то Web-браузер отобразит его в одну строку: Строка1 Строка2 Строка3. Для того чтобы строки располагались друг под другом, необходимо добавить тег
 в конец каждой строки:

```
<p>Строка1<br>
Строка2<br>
Строка3</p>
```

Заметьте, что символ переноса строки в предыдущем результате был заменен символом пробела. Аналогичная ситуация будет, если внутри HTML-кода идут несколько пробельных символов подряд, — все они преобразуются Web-браузером

в один пробел. Если нужно отобразить несколько пробелов подряд, то следует воспользоваться комбинацией :

```
<p>Пункт1<br>
   Пункт1_1<br>
   Пункт1_2</p>
```

Результат в окне Web-браузера:

```
Пункт1
   Пункт1_1
   Пункт1_2
```

Для вывода текста в том же виде, что и в исходном коде, можно воспользоваться парным блочным тегом <pre>:

```
<pre>
Пункт1
   Пункт1_1
   Пункт1_2
</pre>
```

В этом примере строки также будут располагаться друг под другом, и пробелы будут сохранены, но текст выведется моноширинным шрифтом.

1.4.3. Выделение фрагментов текста

Тег <b> отображает текст полужирным шрифтом:

```
<b>Полужирный шрифт</b>
```

Вместо тега <b> можно использовать тег логического форматирования <strong>, который определяет важный фрагмент текста:

```
<strong>Важный фрагмент текста</strong>
```

Тег <i> отображает текст курсивом:

```
<i>Текст, выделенный курсивом</i>
```

Вместо тега <i> можно использовать тег логического форматирования <em>, с помощью которого делается акцент на выделенном фрагменте:

```
<em>Текст, выделенный курсивом</em>
```

Тег <u> отображает текст подчеркнутым:

```
<u>Подчеркнутый текст</u>
```

Тег <s> отображает текст перечеркнутым:

```
<s>Перечеркнутый текст</s>
```

Отметить фрагмент текста как удаленный позволяет парный тег <del>. Текст отображается перечеркнутым. Тег <del> имеет следующие параметры:

- cite — задает ссылку на страницу с описанием причины удаления;
- datetime — дата и время удаления.

Текст, вставленный вместо удаленного, отмечается тегом `<ins>`. Текст отображается подчеркнутым:

```
<del>старый текст</del> <ins>новый текст</ins>
```

Для вывода текста шрифтом меньшего размера применяется парный тег `<small>`:

```
Текст <small>меньшего</small> размера
```

1.4.4. Теги логического форматирования

Теги, рассмотренные в предыдущем разделе, в основном являются тегами физического форматирования. Исключение составляют теги `<strong>` и `<em>`. Эти теги являются *тегами логического форматирования* текста и служат для выделения очень важных и просто важных фрагментов соответственно. Теги логического форматирования используются для структурной разметки документа и могут отображаться разными Web-браузерами по-разному. Приведем основные теги логического форматирования:

- `<cite>...</cite>` — применяется для отметки цитат, а также названий произведений;
- `<code>...</code>` — служит для отметки фрагментов программного кода;
- `<abbr>...</abbr>` — используется для отметки аббревиатур;
- `<kbd>...</kbd>` — отмечает фрагмент как вводимый пользователем с клавиатуры;
- `<q>...</q>` — используется для отметки коротких цитат. Фрагмент отображается в кавычках;
- `<samp>...</samp>` — применяется для отметки результата, выдаваемого программой;
- `<var>...</var>` — отмечает имена переменных;
- `<dfn>...</dfn>` — служит для выделения нового термина (текст отображается курсивом).

В HTML 5 доступны также следующие теги:

- `<mark>...</mark>` — помечает важный фрагмент текста. Браузер Firefox устанавливает для такого фрагмента желтый цвет фона (как бы выделяет фрагмент текста маркером);
- `<time>...</time>` — дата и время. Может включать параметр `datetime`, указывающий дату и время в формате:

  ```
  <год>-<месяц>-<число> <часы>:<минуты>
  ```

1.4.5. Создание нижних и верхних индексов

Тег `<sub>` сдвигает текст ниже уровня строки и уменьшает размер шрифта. Он используется для создания нижних индексов:

```
Формула воды H<sub>2</sub>O
```

Тег <sup> сдвигает текст выше уровня строки и уменьшает размер шрифта. Этот тег используется чаще всего для создания степеней:

```
Единица измерения площади — м<sup>2</sup>
```

1.4.6. Тег *<span>*

Тег <span> является универсальным строчным элементом, к которому можно прикрепить различные стили. Пример использования тега <span> приведен в листинге 1.6.

Листинг 1.6. Тег <span>

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Элемент SPAN</title>
   <style type="text/css">
   span { /* Стиль для всех элементов SPAN */
      font-size: 12pt;       /* Размер шрифта */
      color: blue;           /* Цвет шрифта */
      font-family: "Arial"; /* Название шрифта */
      font-weight: bold;     /* Жирность шрифта */
   }
   </style>
</head>
<body>
  <p>
    С помощью элемента <span>SPAN</span> можно отформатировать
    <span style="color: red">фрагмент</span> внутри абзаца.
  </p>
</body>
</html>
```

1.5. Списки

Список — это набор упорядоченных абзацев текста, помеченных специальными значками (*маркированные списки*) или цифрами (*нумерованные списки*). Рассмотрим каждый из вариантов в отдельности.

1.5.1. Маркированные списки

Маркированный список помещают внутри парного тега <ul>. Перед каждым пунктом списка необходимо поместить тег <li>. Закрывающий тег не обязателен. В листинге 1.7 представлена структура маркированного списка.

Листинг 1.7. Маркированный список

```
<ul>
  <li>Первый пункт</li>
  <li>Второй пункт</li>
</ul>
```

В HTML 4.01 в формате `Transitional` тег `<ul>` имеет параметр `type`, позволяющий задать значок, которым помечаются строки списка. В HTML 5 параметра нет. Вместо него нужно использовать стили:

```
<ul style="list-style-type: circle">
  <li>Первый пункт</li>
  <li>Второй пункт</li>
</ul>
```

1.5.2. Нумерованные списки

Нумерованный список помещают внутри парного тега `<ol>`. Перед каждым пунктом списка необходимо поместить тег `<li>`. Закрывающий тег `</li>` не обязателен.

В листинге 1.8 показана структура нумерованного списка.

Листинг 1.8. Нумерованный список

```
<ol>
   <li>Первый пункт</li>
   <li>Второй пункт</li>
</ol>
```

В HTML 5 тег `<ol>` имеет три параметра. Первый из них — `type` — позволяет задать формат, которым нумеруются пункты списка.

Параметр может принимать следующие значения:

- `A` — пункты нумеруются прописными латинскими буквами:

  ```
  <ol type="A">
     <li>Первый пункт</li>
     <li>Второй пункт</li>
  </ol>
  ```

- `a` — пункты нумеруются строчными латинскими буквами:

  ```
  <ol type="a">
     <li>Первый пункт</li>
     <li>Второй пункт</li>
  </ol>
  ```

- `I` — пункты нумеруются прописными римскими цифрами:

```
<ol type="I">
   <li>Первый пункт</li>
   <li>Второй пункт</li>
</ol>
```

- `i` — пункты нумеруются строчными римскими цифрами:

```
<ol type="i">
   <li>Первый пункт</li>
   <li>Второй пункт</li>
</ol>
```

- `1` — пункты нумеруются арабскими цифрами (по умолчанию):

```
<ol type="1">
   <li>Первый пункт</li>
   <li>Второй пункт</li>
</ol>
```

Второй параметр тега `<ol>` — `start` — задает номер, с которого будет начинаться нумерация пунктов:

```
<ol type="1" start="5">
   <li>Первый пункт</li>
   <li>Второй пункт</li>
</ol>
```

Тег `<li>` также имеет параметр `value`, который позволяет изменить номер указанного элемента списка:

```
<ol>
   <li>Первый пункт</li>
   <li value="5">Второй пункт</li>
   <li>Третий пункт</li>
</ol>
```

В этом примере `Первый пункт` будет иметь номер `1`, `Второй пункт` — номер `5`, а `Третий пункт` — номер `6`.

В HTML 5 тег `<ol>` имеет третий параметр — `reversed` — задающий обратный порядок нумерации:

```
<ol reversed>
   <li>Первый пункт</li>
   <li>Второй пункт</li>
</ol>
```

В этом примере `Первый пункт` будет иметь номер `2`, а `Второй пункт` — номер `1`.

Параметр `reversed` не поддерживается браузером Internet Explorer.

1.5.3. Списки определений

Списки определений состоят из пар «термин/определение». Описываются они с помощью парного тега <dl>. Для вставки термина применяется тег <dt>, а для вставки определения — тег <dd>. Закрывающие теги </dt> и </dd> не обязательны. Пример использования списков определений приведен в листинге 1.9.

Листинг 1.9. Списки определений

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Списки определений</title>
</head>
<body>
  <dl>
    <dt>HTML (HyperText Markup Language)</dt>
      <dd>
         Язык разметки документа, описывающий форму отображения
         информации на экране компьютера
      </dd>
    <dt>CSS (Cascading Style Sheets)</dt>
      <dd>Каскадные таблицы стилей</dd>
  </dl>
</body>
</html>
```

1.6. Графика

Применение графики делает Web-страницу визуально привлекательнее, т. к. изображения помогают лучше передать суть и содержание документа. В Интернете применяются следующие графические форматы:

- GIF — использует только 256 цветов и поддерживает прозрачность. Кроме того, GIF-файл может содержать анимацию;
- JPEG — метод сжатия фотографий с потерей качества. Прозрачность и анимация не поддерживаются;
- PNG — формат хранения графики, использующий сжатие без потерь. Поддерживает прозрачность. Разрабатывался в качестве замены формата GIF.

1.6.1. Изображение на Web-странице

Изображения вставляются в Web-страницы с помощью одинарного тега <img>. Сам тег <img> должен быть расположен внутри блочного тега, например: <p>, <div> или др.

Тег имеет следующие параметры:

- `src` — URL-адрес файла графического изображения:

```
<img src="foto.gif" alt="Текст подсказки">
<img src="http://www.mysite.ru/foto.gif" alt="Текст подсказки">
```

- `alt` — строка текста, которая будет выводиться на месте появления изображения до его загрузки или при отключенной графике, а также если изображение загрузить не удалось:

```
<img src="foto.gif" alt="Текст подсказки">
```

- `width` — ширина изображения в пикселах:

```
<img src="foto.gif" width="480" alt="Текст подсказки">
```

- `height` — высота изображения в пикселах:

```
<img src="foto.gif" width="480" height="60" alt="Подсказка">
```

> ***Примечание***
>
> Значения параметров `width` и `height` могут не соответствовать реальным размерам изображения. В этом случае Web-браузер выполнит перемасштабирование. Если значение одного из параметров указать неправильно, то изображение будет искажено. Если указать только один параметр, то значение второго будет рассчитано пропорционально значению первого, исходя из реальных размеров изображения.

> ***Совет***
>
> Всегда указывайте значения параметров `width` и `height`, т. к. это позволит Web-браузеру отформатировать Web-страницу до загрузки изображений. В противном случае загрузка каждого изображения приведет к необходимости произвести форматирование еще раз, что в свою очередь приведет к перемещению других элементов Web-страницы. В результате картинка в окне Web-браузера будет дергаться.

Следующие параметры доступны только в HTML 5 (браузер Internet Explorer их не поддерживает):

- `srcset` — задает список изображений в формате:

  ```
  <Путь> [<Ширина>w] [<Плотность пикселов>x][, ...]
  ```

  После пути к изображению должна быть указана ширина или плотность пикселов устройства (по умолчанию `1x`). Учитывая эти параметры, Web-браузер выбирает подходящее изображение из списка. Иными словами, для каждого размера экрана мы можем указать разные изображения (например, отличающиеся размерами);

- `sizes` — задает размеры изображения для разных размеров страницы. В качестве единиц измерения могут быть заданы `em`, `ex`, `ch`, `rem`, `vw` (ширина в процентах), `vh` (высота в процентах), `vmin`, `vmax`, `cm`, `mm`, `q`, `in`, `pc`, `pt`, `px` (пикселы). В качестве значения можно использовать медиазапросы (*см. разд. 2.24*).

Пример использования параметров `srcset` и `sizes`:

```
<img src="img_large.jpg" alt="Подсказка"
   srcset="img_small.jpg 500w, img_medium.jpg 750w, img_large.jpg 1000w"
   sizes="100vw" style="max-width: 100%">
```

Если ширина экрана окажется меньше или равна `500 px`, будет использовано изображение `img_small.jpg`, если меньше или равна `750 px` — `img_medium.jpg`, в противном случае — `img_large.jpg`. Если Web-браузер не поддерживает параметры `srcset` и `sizes`, то будет использоваться изображение, указанное в параметре `src`.

1.6.2. Изображение в качестве фона

В HTML 4.01 в формате `Transitional` тег `<body>` имеет параметры `background` (позволяет задать фоновый рисунок для документа) и `bgcolor` (цвет фона). В параметре `bgcolor` следует указывать цвет, близкий к цвету фонового изображения, т. к. резкий переход, например, от светлого тона к темному вызовет неприятное мелькание, — ведь фоновое изображение может загрузиться с некоторой задержкой.

В HTML 5 эти параметры отсутствуют, вместо них следует использовать стили:

```
<body style="background: gray url(foto.gif) repeat">
```

1.6.3. Тег *<picture>*

В HTML 5 был добавлен парный тег `<picture>`, внутри которого с помощью тега `<source>` можно задать различные изображения для разных размеров экрана.

Тег `<source>` имеет следующие параметры:

- `srcset` — задает список изображений в формате:

  ```
  <Путь> [<Ширина>w] [<Плотность пикселов>x][, ...]
  ```

  После пути к изображению должна быть указана ширина или плотность пикселов устройства (по умолчанию `1x`). Учитывая эти параметры, Web-браузер выбирает подходящее изображение из списка. Иными словами, для каждого размера экрана мы можем указать разные изображения (например, отличающиеся размерами);
- `sizes` — задает размеры изображения для разных размеров страницы. В качестве единиц измерения могут быть заданы `em`, `ex`, `ch`, `rem`, `vw` (ширина в процентах), `vh` (высота в процентах), `vmin`, `vmax`, `cm`, `mm`, `q`, `in`, `pc`, `pt`, `px` (пикселы);
- `media` — позволяет указать медиазапросы (*см. разд. 2.24*).

Пример:

```
<picture style="max-width: 100%">
<source sizes="100vw"
srcset="img_small.jpg 500w, img_medium.jpg 750w, img_large.jpg 1000w">
<img src="img_large.jpg" alt="Подсказка">
</picture>
```

Если Web-браузер не поддерживает тег `<picture>`, то будет отображено изображение, указанное в параметре `src` тега `<img>`.

1.6.4. SVG-графика

В HTML 5 помимо растровых изображений можно загрузить и отобразить изображение в формате SVG. Это векторный формат, описывающий изображение как набор примитивов: прямых, кривых, многоугольников и др. (полный список примитивов и их подробное описание смотрите на странице **https://www.w3.org/TR/SVG/**). Векторные изображения масштабируются без потери качества, а формат SVG, кроме того, поддерживает создание анимированных изображений и даже интерактивной графики, откликающейся на действия пользователя.

Изображения в формате SVG могут быть вставлены на страницу с помощью тега `<img>`, но лучше для этого использовать тег `<picture>` и внутри него с помощью тега `<img>` задать альтернативное изображение в растровом формате для Web-браузеров, не поддерживающих формат SVG:

```
<picture>
   <source srcset="img.svg">
   <img src="photo.jpg" alt="Подсказка">
</picture>
```

Содержимое файла img.svg:

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <rect x="1" y="1" width="1198" height="398"
        fill="none" stroke="red" stroke-width="2" />
  <rect x="400" y="100" width="200" height="200" rx="50"
        fill="green" stroke="black" stroke-width="5" />
  <circle cx="200" cy="200" r="100" fill="red" />
  <polygon fill="yellow" stroke="blue" stroke-width="3"
           points="850,75  958,137.5 958,262.5
                   850,325 742,262.6 742,137.5" />
</svg>
```

В HTML 5 существует также тег `<svg>`, с помощью которого можно формировать векторное изображение прямо в HTML-документе:

```
<svg width="12cm" height="4cm" viewBox="0 0 1200 400"
     xmlns="http://www.w3.org/2000/svg" version="1.1">
  <circle cx="200" cy="200" r="100" fill="red" />
</svg>
```

1.6.5. Тег <*canvas*>

В HTML 5 появилась поддержка программного рисования произвольной растровой графики на холсте. Холст создается с помощью парного тега `<canvas>`, имеющего следующие параметры:

- `width` — ширина холста;
- `height` — высота холста.

Пример:

```
<canvas id="cnv" width="400" height="300">
<p>Ваш Web-браузер не поддерживает элемент canvas</p>
</canvas>
```

Если параметры не указаны, то создается холст с размерами 300×150 пикселов.

> **ПРИМЕЧАНИЕ**
> Подробнее о программном рисовании на холсте будет рассказано в *главе 3*.

1.7. Гиперссылки

Гиперссылки позволяют нажатием кнопки мыши быстро перемещаться от одного документа к другому. Именно гиперссылки связывают все Web-страницы Интернета в единую сеть.

1.7.1. Внешние гиперссылки

Внешние гиперссылки вставляются в HTML-документ с помощью тега `<a>`. Сам тег `<a>` должен быть расположен внутри блочного тега, например: `<p>`, `<div>` или др.

Основным параметром тега `<a>` является `href`. Именно этот параметр задает URL-адрес Web-страницы, которая будет загружена при щелчке мыши на указателе. В качестве указателя может быть текст:

```
<a href="http://www.mysite.ru/file.html">Текст ссылки</a>
```

или изображение:

```
<a href="http://www.mysite.ru/file.html">
<img src="foto.gif" alt="Подсказка"></a>
```

Кроме HTML-документов, можно ссылаться и на файлы других типов — например, изображения, архивы и т. д. По умолчанию выполняется переход по ссылке, и содержимое документа загружается в окно Web-браузера. Если невозможно отобразить документ в окне, то Web-браузер предложит сохранить файл. В HTML 5 тег `<a>` имеет параметр `download`, который говорит Web-браузеру, что документ нужно скачать, а не открыть. В качестве значения параметр `download` может содержать предлагаемое имя файла для сохранения. MIME-тип документа следует указать в параметре `type`:

```
<a href="foto.gif">Открыть</a>
<a href="foto.gif" download type="image/gif">Скачать</a>
<a href="foto.gif" download="test.gif" type="image/gif">
   Скачать и сохранить под названием test.gif</a>
```

Если URL-адрес содержит символ `&`, то его необходимо заменить HTML-эквивалентом `&`:

```
<a href="index.php?id=5&amp;name=Nik">Текст ссылки</a>
```

URL-адреса бывают *абсолютными* и *относительными*.

Абсолютный URL-адрес

Абсолютный URL-адрес содержит обозначение протокола, доменный или IP-адрес компьютера, путь к файлу, а также имя файла:

```
http://www.mysite.ru/folder/file.html
```

Если файл находится в корневой папке, то путь может отсутствовать:

```
http://www.mysite.ru/file.html
```

Имя файла также может отсутствовать. В этом случае загружается Web-страница, заданная по умолчанию в настройках Web-сервера (чаще всего index.html или index.php):

```
http://www.mysite.ru/
http://www.mysite.ru/folder/
```

Относительный URL-адрес

При относительном задании URL-адреса путь определяется с учетом местоположения Web-страницы, на которой находится ссылка. Возможны следующие варианты:

- если нужная Web-страница находится в той же папке, что и Web-страница, содержащая ссылку, то URL-адрес может содержать только имя файла. Если с Web-страницы, находящейся по адресу **http://www.mysite.ru/folder1/folder2/file1.html**, нужно перейти на Web-страницу **http://www.mysite.ru/folder1/folder2/file2.html**, то ссылка будет такой:

  ```
  <a href="file2.html">Текст ссылки</a>
  ```

- если с Web-страницы, находящейся по адресу **http://www.mysite.ru/folder1/folder2/file1.html**, нужно перейти на Web-страницу **http://www.mysite.ru/folder1/folder2/folder3/file2.html**, то ссылку можно указать так:

  ```
  <a href="folder3/file2.html">Текст ссылки</a>
  ```

- если с Web-страницы, находящейся по адресу **http://www.mysite.ru/folder1/folder2/file1.html**, нужно перейти на Web-страницу **http://www.mysite.ru/folder1/file2.html**, то ссылка будет такой:

  ```
  <a href="../file2.html">Текст ссылки</a>
  ```

А при переходе с **http://www.mysite.ru/folder1/folder2/folder3/file1.html** на **http://www.mysite.ru/folder1/file2.html** — такой:

```
<a href="../../file2.html">Текст ссылки</a>
```

- если в разделе HEAD присутствует тег <base> с параметром href, то все относительные ссылки разрешаются от указанного в параметре значения, а не от текущего местоположения страницы:

```
<base href="http://www.mysite.ru/folder1/">
```

- если первым символом в пути указан символ /, то путь определяется относительно корня сайта.

Очень часто необходимо загрузить документ в новую вкладку, а не в текущую. Для этого в параметре target тега <a> следует указать значение _blank:

```
<a href="http://www.mysite.ru/file.html" target="_blank">Ссылка</a>
```

Если нужно, чтобы все ссылки открывались на новой вкладке, то в раздел HEAD нужно добавить следующий код:

```
<base target="_blank">
```

Другие значения параметра target мы рассмотрим при изучении фреймов (см. *разд. 1.9.2*).

1.7.2. Внутренние гиперссылки

С помощью внутренних гиперссылок можно создать ссылки на разные разделы текущей Web-страницы. Если документ очень большой, то наличие внутренних гиперссылок позволяет быстро перемещаться между разделами.

Внутренняя гиперссылка также вставляется при помощи тега <a> с одним различием — параметр href содержит имя «якоря», а не URL-адрес. Перед именем «якоря» ставится знак #:

```
<a href="#chapter1">Глава 1</a>
```

Также можно сослаться на «якорь» другого документа. Это делается так:

```
<a href="http://www.mysite.ru/file.html#chapter1">Текст</a>
```

«Якорь» создается с помощью параметра id, задающего уникальный идентификатор элемента. Этот параметр имеют практически все теги.

Структура документа с внутренними ссылками приведена в листинге 1.10.

Листинг 1.10. Структура документа с внутренними ссылками

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
```

```
   <title>Создание внутренних ссылок</title>
   <style type="text/css">
   p { margin-bottom: 150px }
   </style>
</head>
<body>
   <h1>Название документа</h1>
   <h2>Оглавление</h2>
   <ul>
      <li><a href="#chapter1">Глава 1</a>
      <li><a href="#chapter2">Глава 2</a>
      <li><a href="#chapter3">Глава 3</a>
      <li><a href="#chapter4">Глава 4</a>
   </ul>
   <h2 id="chapter1">Глава 1</h2>
   <p>Содержание главы 1</p>
   <h2 id="chapter2">Глава 2</h2>
   <p>Содержание главы 2</p>
   <h2 id="chapter3">Глава 3</h2>
   <p>Содержание главы 3</p>
   <h2 id="chapter4">Глава 4</h2>
   <p>Содержание главы 4</p>
</body>
</html>
```

1.7.3. Гиперссылки на адрес электронной почты

Ссылка на адрес электронной почты выглядит так:

```
<a href="mailto:mail@mysite.ru">Текст</a>
```

Вместо URL-адреса указывается адрес электронной почты, перед которым добавляется слово `mailto:`.

> **Совет**
>
> Не следует публиковать ссылку с адресом электронной почты на сайте. Такие ссылки автоматически собираются роботами, и в дальнейшем этот E-mail будет завален спамом.

1.8. Таблицы

В HTML-документе таблицы используются в следующих случаях:

- как средство представления данных;
- как элемент оформления страницы, с помощью которого можно точно разместить на странице текст и графику.

Начнем со структуры, описывающей таблицу (листинг 1.11).

Листинг 1.11. Структура HTML-таблиц

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Структура HTML-таблиц</title>
   <style type="text/css">
   table {
      width: 300px;                /* Ширина таблицы */
      border: black 1px solid;     /* Стиль рамки таблицы */
      border-spacing: 0;           /* Расстояние между ячейками */
      border-collapse: collapse;   /* Одинарная рамка */
      margin: 0 auto 0 auto        /* Выравнивание таблицы по центру */
   }
   caption {
      text-align: center;          /* Выравнивание заголовка по центру */
      caption-side: top            /* Вывод заголовка над таблицей */
   }
   td, th {
      border: black 1px solid;     /* Стиль рамки ячеек */
      padding: 10px;            /* Отступ между границей и содержимым */
      text-align: center           /* Выравнивание ячейки по центру */
   }
   tr {
      text-align: center;          /* Горизонтальное выравнивание */
      vertical-align: top;         /* Вертикальное выравнивание */
   }
   tr:nth-child(2n) {
      background: #e8e8e8          /* Зебра */
   }
   thead, tfoot {
      background: #f5e0cd          /* Цвет фона */
   }
   </style>
</head>
<body>
   <table>
      <caption>Заголовок таблицы</caption>
      <thead>
      <tr>
         <th>Столбец 1</th>
         <th>Столбец 2</th>
      </tr>
      </thead>
      <tbody>
```

```
      <tr>
         <td>Ячейка 1</td>
         <td>Ячейка 2</td>
      </tr>
      <tr>
         <td>Ячейка 3</td>
         <td>Ячейка 4</td>
      </tr>
      </tbody>
      <tfoot>
      <tr>
         <th>Итого 1</th>
         <th>Итого 2</th>
      </tr>
      </tfoot>
   </table>
</body>
</html>
```

Эта структура описывает таблицу 2×2 с заголовком, заголовками столбцов и строкой итогов. Значения в ячейках выровнены по центру. Все ячейки таблицы пронумерованы от 1 до 4.

Таблица вставляется в HTML-документ с помощью парного тега `<table>`. Отдельная ячейка таблицы описывается тегом `<td>`, а ряд ячеек — с помощью тега `<tr>`. Ячейка заголовка описывается с помощью тега `<th>`. Тег `<caption>` позволяет задать заголовок таблицы.

Для логического форматирования таблицы предназначены теги `<thead>`, `<tbody>` и `<tfoot>`. Тег `<thead>` описывает заголовок столбцов таблицы, тег `<tbody>` — основное содержимое таблицы, а тег `<tfoot>` — «подвал» таблицы.

Закрывающие теги `</thead>`, `</tbody>`, `</tfoot>`, `</td>`, `</th>` и `</tr>` не обязательны.

1.8.1. Вставка таблицы в документ

В HTML 4.01 тег `<table>` имеет параметры `border` (управляет отображением линий сетки таблицы, а также задает толщину рамки вокруг таблицы. По умолчанию сетка не отображается), `cellspacing` (задает толщину линий сетки внутри таблицы, точнее сказать, расстояние между рамками соседних ячеек), `cellpadding` (указывает размер отступа между рамкой ячейки и данными внутри ячейки), `width` (определяет ширину таблицы), `align` (задает выравнивание таблицы, а также обтекание таблицы текстом) и `bgcolor` (указывает цвет фона таблицы).

В HTML 5 этих параметров нет, вместо них следует использовать стили:

```
table {
   width: 300px;              /* Ширина таблицы
                                 (вместо параметра width) */
```

```
   border: black 1px solid;     /* Стиль рамки таблицы
                                   (вместо параметра border) */
   border-spacing: 0;           /* Расстояние между ячейками
                                   (вместо параметра cellspacing) */
   border-collapse: collapse;   /* Одинарная рамка */
   margin: 0 auto 0 auto        /* Выравнивание таблицы по центру
                                   (вместо параметра align) */
}
td, th {
   border: black 1px solid;     /* Стиль рамки ячеек */
   padding: 10px;               /* Отступ между границей и содержимым
                                   (вместо параметра cellpadding) */
}
```

1.8.2. Заголовок таблицы

Тег `<caption>` позволяет задать заголовок таблицы. В HTML 4.01 в формате `Transitional` тег `<caption>` имеет параметр `align`, задающий положение заголовка. В HTML 5 этого параметра нет, вместо него следует использовать стили:

```
caption {
   text-align: center;          /* Выравнивание заголовка по центру */
   caption-side: top            /* Вывод заголовка над таблицей */
}
```

Атрибут стиля `caption-side` (замена параметра `align`) может принимать следующие значения:

- `top` — заголовок помещается над таблицей (значение по умолчанию);
- `bottom` — заголовок располагается под таблицей.

1.8.3. Строки таблицы

Строки таблицы описываются с помощью парного тега `<tr>`. Закрывающий тег не обязателен. В HTML 4.01 тег `<tr>` имеет параметры `align` (горизонтальное выравнивание текста в ячейках таблицы), `valign` (вертикальное выравнивание текста в ячейках таблицы) и `bgcolor` (цвет фона ячеек таблицы). В HTML 5 этих параметров нет, вместо них следует использовать стили:

```
tr {
   text-align: center;          /* Горизонтальное выравнивание */
   vertical-align: top;         /* Вертикальное выравнивание */
}
```

Чтобы таблица лучше читалась, цвет фона соседних строк нужно сделать отличающимся. Выполнить такое чередование позволяет следующее правило стиля:

```
tr:nth-child(2n) {
   background: #e8e8e8          /* Зебра */
}
```

1.8.4. Столбцы таблицы

По умолчанию таблица отображается только после полной загрузки. Если нужно отображать данные построчно, то следует задать характеристики столбцов с помощью тегов <col> (описывает один столбец) и <colgroup> (описывает группу столбцов). Теги имеют параметр span, который задает число колонок.

Характеристики колонок в HTML 5 задаются с помощью стилей:

```
<table>
   <colgroup style="background: #f5e0cd">
      <col style="width: 300px">
      <col style="width: 100px">
   </colgroup>
   <tr>
      <td>Ячейка 1</td><td>Ячейка 2</td>
   </tr>
   <tr>
      <td>Ячейка 3</td><td>Ячейка 4</td>
   </tr>
</table>
```

1.8.5. Ячейки таблицы

Ячейки таблицы описываются с помощью тегов <td> (обычная ячейка) и <th> (ячейка заголовка — содержимое таких ячеек выделяется полужирным шрифтом и выравнивается по центру). Закрывающие теги не обязательны.

Теги <td> и <th> имеют следующие параметры:

- colspan — задает количество объединяемых ячеек по горизонтали;
- rowspan — указывает количество объединяемых ячеек по вертикали.

В качестве примера объединим горизонтально расположенные ячейки в одну (листинг 1.12).

Листинг 1.12. Объединение ячеек по горизонтали

```
<table>
   <tr>
      <td colspan="2">Ячейки 1 и 2 объединены</td>
   </tr>
   <tr>
      <td>Ячейка 3</td>
      <td>Ячейка 4</td>
   </tr>
</table>
```

Теперь объединим вертикально расположенные ячейки 1 и 3 в одну (листинг 1.13).

Листинг 1.13. Объединение ячеек по вертикали

```
<table>
   <tr>
      <td rowspan="2">Ячейки 1 и 3 объединены</td>
      <td>Ячейка 2</td>
   </tr>
   <tr>
      <td>Ячейка 4</td>
   </tr>
</table>
```

Если для ячейки нет значения, то вместо значения следует вставить неразрывный пробел:

```
<td> </td>
```

1.9. Фреймы

Фреймы позволяют разбить окно Web-браузера на несколько прямоугольных областей, в каждую из которых можно загрузить отдельный HTML-документ.

В HTML 4.01 в формате `Frameset` поддерживаются теги `<frameset>`, `<frame>` и `<noframes>`. В HTML 5 эти теги отсутствуют, вместо них нужно использовать тег `<iframe>`.

1.9.1. Тег *<iframe>*. Добавление фрейма в обычный документ

Вставлять фреймы в обычный HTML-документ можно с помощью парного тега `<iframe>` . Иногда такие фреймы называют «плавающими».

Тег `<iframe>` имеет следующие параметры:

- `src` — определяет URL-адрес документа, который должен быть загружен во фрейм. Может быть указан абсолютный или относительный URL-адрес:

  ```
  <iframe src="chapter1.html"></iframe>
  ```

- `name` — задает уникальное имя фрейма:

  ```
  <iframe src="chapter1.html" name="myframe"></iframe>
  ```

- `width` и `height` — задают ширину и высоту фрейма соответственно:

  ```
  <iframe src="chapter1.html" name="myframe" width="700"
          height="400"></iframe>
  ```

В HTML 5 добавлены следующие параметры:

- `srcdoc` — позволяет задать HTML-код, который отобразится во фрейме (не поддерживается браузером Internet Explorer):

  ```
  <iframe srcdoc="<p>Текст внутри фрейма"></iframe>
  ```

- `sandbox` — устанавливает ограничения на содержимое, загружаемое во фрейм. Если параметр указан без значения, то устанавливается максимальный режим ограничения. Можно снять отдельные ограничения, указав следующие значения через пробел:
  - `allow-forms` — разрешает отправку форм;
  - `allow-modals` — разрешает открытие модальных диалоговых окон;
  - `allow-popups` — разрешает всплывающие окна;
  - `allow-scripts` — разрешает выполнение скриптов (исключая всплывающие окна);
  - `allow-top-navigation` — разрешает загружать документы по ссылкам в родительский документ;
  - `allow-same-origin` — разрешает документу внутри фрейма из того же источника доступ к родительскому документу;
  - `allow-pointer-lock` — разрешает Pointer Lock API;
  - `allow-presentation` — разрешает Presentation API;
  - `allow-orientation-lock` — разрешает Screen Orientation API.

Попробуем создать Web-страницу с плавающим фреймом. Для этого создадим в одной папке три файла:

- chapter1.html (листинг 1.14) — содержание главы 1;
- chapter2.html (листинг 1.15) — содержание главы 2;
- index.html (листинг 1.16) — HTML-документ с плавающим фреймом.

Листинг 1.14. Содержимое файла chapter1.html

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Глава 1</title>
</head>
<body>
   <h1>Глава 1</h1>
   <p>Содержание главы 1</p>
</body>
</html>
```

Листинг 1.15. Содержимое файла chapter2.html

```
<!DOCTYPE html>
<html lang="ru">
```

```
<head>
   <meta charset="utf-8">
   <title>Глава 2</title>
</head>
<body>
   <h1>Глава 2</h1>
   <p>Содержание главы 2</p>
</body>
</html>
```

Листинг 1.16. Применение плавающих фреймов

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Применение плавающих фреймов</title>
</head>
<body>
  <h1>Название документа</h1>
  <h2>Оглавление</h2>
  <ul>
    <li><a href="chapter1.html" target="chapter">Глава 1</a></li>
    <li><a href="chapter2.html" target="chapter">Глава 2</a></li>
  </ul>
  <iframe src="chapter1.html" name="chapter" width="700"
          height="400"></iframe>
</body>
</html>
```

Откроем в Web-браузере файл index.html. При переходе по ссылкам соответствующая страница загружается в окно фрейма, а заголовок и панель навигации остаются в неизменном состоянии.

> ***Примечание***
>
> При использовании фреймов следует учитывать, что поисковые машины при индексации сайтов заносят в свои базы именно отдельные страницы структуры фреймов, а не саму структуру. Это обстоятельство полностью разрушает всю структуру сайта. Ведь если панель навигации расположена на одной странице, а основная часть страницы на другой, то при переходе посетителя с поискового портала он попадает сразу на основную часть, а панель навигации остается ему не доступна.

1.9.2. Загрузка документа в определенный фрейм

Для загрузки документа в определенный фрейм существует параметр `target` тега `<a>`. В параметре `target` указывается имя фрейма (которое задается с помощью параметра `name` тега `<iframe>`) или одно из зарезервированных значений:

- _blank — документ будет загружен в новую вкладку (если в теге <iframe> указан параметр sandbox и не заданы значения allow-same-origin и allow-popups, то ссылка работать не будет):

  ```
  <a href="file1.html" target="_blank">Текст ссылки</a>
  ```

- _self — документ будет загружен в тот же фрейм (или вкладку), где находится гиперссылка (если в теге <iframe> указан параметр sandbox и не задано значение allow-same-origin, то ссылка работать не будет):

  ```
  <a href="file1.html" target="_self">Текст ссылки</a>
  ```

- _parent — документ будет загружен в окне, являющемся родительским по отношению к текущему фрейму (если в теге <iframe> указан параметр sandbox и не заданы значения allow-same-origin и allow-top-navigation, то ссылка работать не будет):

  ```
  <a href="file1.html" target="_parent">Текст ссылки</a>
  ```

- _top — документ будет загружен поверх всех фреймов (если в теге <iframe> указан параметр sandbox и не заданы значения allow-same-origin и allow-top-navigation, то ссылка работать не будет):

  ```
  <a href="file1.html" target="_top">Текст ссылки</a>
  ```

Если нужно загрузить документ во фрейм с именем chapter, то ссылка будет такой:

```
<a href="chapter1.html" target="chapter">Глава 1</a>
```

Имя фрейма задается с помощью параметра name тега <iframe>:

```
<iframe src="chapter1.html" name="chapter" width="700"
        height="400"></iframe>
```

1.10. Карты-изображения

С помощью *карт-изображений* можно создать очень красивую панель навигации. В качестве ссылок на разделы сайта будут служить области на обычном изображении формата GIF, JPEG или PNG.

К минусам такого подхода можно отнести:

- Web-браузеры не смогут выделять другим цветом уже пройденные ссылки;
- поскольку вместо текстовых ссылок используются изображения, это приведет к увеличению времени загрузки страницы;
- пользователи могут отключить использование графики и по этой причине не увидят панель навигации.

1.10.1. Карта-изображение как панель навигации

Давайте перепишем файл index.html (мы использовали его при изучении плавающих фреймов в листинге 1.16) и заменим текстовую панель навигации на карту-изображение (листинг 1.17).

Листинг 1.17. Применение карт-изображений

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Применение карт-изображений</title>
</head>
<body>
  <h1>Название документа</h1>
  <h2>Оглавление</h2>
  <p><img src="foto.gif" alt="Подсказка" width="240" height="120"
       usemap="#karta" style="border: black 2px solid">
  <map name="karta">
    <area shape="rect" coords="0,0,120,120" href="chapter1.html"
          target="chapter" alt="Глава 1" title="Глава 1">
    <area shape="rect" coords="120,0,240,120" href="chapter2.html"
          target="chapter" alt="Глава 2" title="Глава 2">
  </map></p>
  <iframe src="chapter1.html" name="chapter" width="700"
          height="400"></iframe>
</body>
</html>
```

Нас сейчас не интересует само изображение, поэтому его может в папке и не быть. Чтобы видеть границы изображения на Web-странице, мы добавили атрибут стиля `border` в параметре `style`. Сохраним файл и обновим Web-страницу.

Итак, как и в предыдущем примере, у нас есть заголовок и окно фрейма, но вместо текстовой панели навигации имеется изображение 240×120 (в нашем примере показана только его рамка). Изображение виртуально разделено пополам на левую и правую части. Если навести курсор мыши на правую часть изображения, то форма курсора даст понять, что это ссылка, а рядом с курсором появится всплывающая подсказка `Глава 2` (значение параметра `title`). При переходе по ссылке файл `chapter2.html` загружается в окно фрейма. Если щелкнуть на левой части изображения, то во фрейм опять вернется текст `Глава 1`.

1.10.2. Структура карт-изображений

Рассмотрим структуру, которая позволяет вставить карту-изображение в HTML-документ (листинг 1.18).

Листинг 1.18. Структура карт-изображений

```
<!-- Часть 1 -->
<img src="foto.gif" alt="Подсказка" width="240" height="120"
       usemap="#karta" style="border: black 2px solid">
```

```
<!-- Часть 1 -->
<!-- Часть 2 -->
<map name="karta">
   <area shape="rect" coords="0,0,120,120" href="chapter1.html"
         target="chapter" alt="Глава 1" title="Глава 1">
   <area shape="rect" coords="120,0,240,120" href="chapter2.html"
         target="chapter" alt="Глава 2" title="Глава 2">
</map>
<!-- Часть 2 -->
```

Как видим, код для вставки карты-изображения состоит из двух частей:

- первая часть с помощью тега <img> вставляет изображение в Web-страницу. Параметр usemap указывает, что изображение является картой. В качестве значения параметра указывается URL-адрес описания конфигурации. Если описание карты расположено в том же HTML-документе, то указывается название раздела конфигурации карты, перед которым добавляется символ #;
- вторая часть, являющаяся описанием конфигурации карты, расположена между тегами <map> и </map>. Активная область карты описывается с помощью тега <area>. Сколько на карте активных областей, столько и тегов <area> должно быть.

1.10.3. Тег *<map>*

Парный тег <map> служит для описания конфигурации областей карты-изображения. У тега <map> есть единственный параметр — name. Значение параметра name должно соответствовать имени в параметре usemap тега <img>.

1.10.4. Описание активной области на карте-изображении

Тег <area> описывает одну активную область на карте. Закрывающий тег не требуется. Если одна активная область перекрывает другую, то будет реализована первая ссылка из списка областей.

Тег <area> имеет следующие параметры:

- shape — задает форму активной области. Он может принимать 4 значения:
  - rect — активная область в форме прямоугольника (по умолчанию);
  - circle — активная область в форме круга;
  - poly — активная область в форме многоугольника;
  - default — активная область занимает всю площадь изображения:

```
<area shape="default" alt="Подсказка"
     href="chapter0.html">
```

- coords — определяет координаты точек отдельной активной области. Координаты указываются относительно левого верхнего угла изображения и задаются следующим образом:
  - для области типа rect задаются координаты верхнего левого и правого нижнего углов прямоугольника (координаты указываются через запятую):

    ```
    coords="x1, y1, x2, y2"
    ```

    Здесь x1 и y1 — координаты левого верхнего угла, а x2 и y2 — координаты правого нижнего угла, например:

    ```
    <area shape="rect" coords="0,0,120,120" alt=""
          href="chapter1.html">
    ```

  - для области типа circle указываются координаты центра круга и радиус:

    ```
    coords="x1, y1, r"
    ```

    Здесь x1 и y1 — координаты центра круга, а r — радиус круга, например:

    ```
    <area shape="circle" coords="60,60,30" alt=""
          href="chapter1.html">
    ```

  - для области типа poly перечисляются координаты вершин многоугольника в нужном порядке:

    ```
    coords="x1, y1, x2, y2, x3, y3"
    ```

    Здесь x1, y1, x2, y2, x3, y3 — координаты вершин многоугольника (в данном случае — треугольника). Можно задавать и большее количество вершин — иными словами, можно описать активную область практически любой формы. Координаты последней вершины не обязательно должны совпадать с первой:

    ```
    <area shape="poly" coords="10,100,60,10,100,100"
          alt="" href="chapter1.html">
    ```

- href — указывает URL-адрес ссылки. Может быть указан абсолютный или относительный адрес ссылки:

  ```
  <area shape="circle" coords="60,60,30" alt=""
        href="chapter1.html">
  ```

- alt — задает альтернативный текст (с помощью параметра title можно задать текст всплывающей подсказки при наведении курсора мыши на активную область):

  ```
  <area shape="rect" coords="0,0,120,120" href="chapter1.html"
        target="chapter" alt="Глава 1" title="Глава 1">
  ```

- target — указывает, куда будет загружен документ при переходе по ссылке. Может быть указано имя фрейма или одно из зарезервированных значений: _blank, _self, _parent или _top. Эти значения рассматривались нами при изучении фреймов (см. *разд. 1.9.2*).

1.11. Формы

Формы предназначены для пересылки данных от пользователя к Web-серверу. О том, как обрабатывать эти данные на стороне сервера, будет рассказано при изучении языка PHP. А пока рассмотрим возможности HTML для создания форм.

1.11.1. Создание формы для регистрации сайта

Создадим форму регистрации сайтов в каталоге ресурсов Интернета. Откроем Notepad++ и наберем код, приведенный в листинге 1.19.

Листинг 1.19. Пример использования форм

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Пример использования форм</title>
</head>
<body>
 <h1>Пример формы регистрации сайта</h1>
 <form action="file.php" method="POST" enctype="multipart/form-data">
  <div>
   Логин:<br>
   <input type="text" name="login"><br>
   Пароль:<br>
   <input type="password" name="passw" value="Пароль"><br>
   URL-адрес сайта:<br>
   <input type="text" name="url" value="http://" size="20"><br>
   Название сайта:<br>
   <input type="text" name="name_site" size="20"><br>
   Описание сайта:<br>
   <textarea name="descr" rows="10" cols="15"></textarea><br>
   Тема сайта:<br>
   <select name="theme">
     <option value="0" selected>Тема не выбрана</option>
     <option value="1">Тема1</option>
     <option value="2">Тема2</option>
     <option value="3">Тема3</option>
   </select><br>
   Баннер 88*31:<br>
   <input type="file" name="banner" size="20"><br><br>
   <input type="reset" value="Очистить">
   <input type="submit" value="Отправить">
  </div>
 </form>
</body>
</html>
```

Сохраним файл под именем form.html и откроем его в Web-браузере — в окне Web-браузера будет отображена форма, состоящая из пяти текстовых полей: **Логин**, **Пароль**, **URL-адрес сайта**, **Название сайта** и **Описание сайта**, списка значений (**Тема сайта**), поля выбора файла с кнопкой **Обзор** (под надписью **Баннер 88*31:**), а также двух кнопок: **Очистить** и **Отправить**.

Кнопка **Очистить** возвращает все значения формы к первоначальным. Кнопка **Отправить** позволяет отправить данные, введенные пользователем, расположенной на Web-сервере программе, URL-адрес которой указан в параметре `action` тега `<form>` (в нашем случае это `file.php`). Программа обработает данные и либо добавит сайт в каталог и выдаст подтверждение об успешной регистрации, либо выдаст сообщение об ошибке, если обязательное поле не заполнено или заполнено неправильно. В нашем случае программы обработки нет, и отправка данных формы ни к чему не приведет — точнее, приведет к ошибке «Файл не найден».

1.11.2. Структура документа с формами

Форма добавляется в HTML-документ при помощи парного тега `<form>`. Внутри тегов `<form>` и `</form>` могут располагаться теги `<input>`, `<textarea>` и `<select>`, вставляющие в форму элементы управления:

```
<form action="file.php">
<input type="text">
<textarea></textarea>
<select>
<option></option>
</select>
<input type="file">
<input type="reset">
<input type="submit">
</form>
```

Рассмотрим эти теги подробно.

1.11.3. Добавление формы в документ

Парный тег `<form>` позволяет добавить форму в HTML-документ. Тег имеет следующие параметры:

- `action` — задает URL-адрес программы обработки формы. Может задаваться в абсолютной или относительной форме:

  ```
  <form action="file.php">
  ```

- `method` — определяет, как будут пересылаться данные из формы Web-серверу. Может принимать два значения: `GET` и `POST`:

  - `GET` — данные формы пересылаются путем их добавления к URL-адресу после знака `?` в формате:

    ```
    [Имя параметра]=[Значение параметра]
    ```

Пары `параметр=значение` отделяются друг от друга символом амперсанда (`&`):

```
http://www.mysite.ru/file.php?login=Login&passw=Password
```

Все специальные символы, а также буквы, отличные от латинских (например, буквы русского языка), кодируются в формате `%nn`, а пробел заменяется знаком +.

В теге `<form>` значение `GET` для параметра `method` задается так:

```
<form action="http://www.mysite.ru/file.php" method="GET">
```

Метод `GET` применяется, когда объем пересылаемых данных невелик, т. к. существует предел длины URL-адреса;

- `POST` — предназначен для пересылки данных большого объема, файлов и конфиденциальной информации (например, паролей):

```
<form action="http://www.mysite.ru/file.php" method="POST">
```

- `enctype` — задает способ кодирования передаваемых данных. Может принимать два значения:
  - `application/x-www-form-urlencoded` — применяется по умолчанию:

```
<form action="http://www.mysite.ru/file.php" method="POST"
enctype="application/x-www-form-urlencoded">
```

  - `multipart/form-data` — указывается при пересылке Web-серверу файлов:

```
<form action="http://www.mysite.ru/file.php" method="POST"
enctype="multipart/form-data">
```

- `name` — задает имя формы:

```
<form action="file.php" name="form1">
```

- `target` — указывает, куда будет помещен документ, являющийся результатом обработки данных формы Web-сервером. Параметр может содержать имя фрейма или одно из зарезервированных значений: `_blank`, `_top`, `_self` или `_parent`:

```
<form action="http://www.mysite.ru/file.php" method="POST"
enctype="multipart/form-data" target="_blank">
```

Эти значения рассматривались нами при изучении фреймов.

В HTML 5 тег `<form>` получил поддержку двух новых параметров:

- `autocomplete` — включает или отключает автодополнение значений, занесенных в поля ввода формы. Значение `on` включает автодополнение, `off` — отключает;
- `novalidate` — если присутствует, то значения, занесенные в поля ввода, не будут проверяться:

```
<form action="process.php" autocomplete="off" novalidate> ...
</form>
```

Браузер Safari этот параметр не поддерживает.

1.11.4. Тег *<input>*

Тег `<input>` позволяет вставить в форму элементы управления, например: текстовое поле, кнопку или флажок. Этот тег имеет следующие параметры:

- `type` — задает тип элемента. В зависимости от значения этого поля создаются следующие элементы формы:
  - `text` — текстовое поле ввода:

    ```
    <input type="text">
    ```

  - `password` — текстовое поле для ввода пароля, в котором все вводимые символы заменяются звездочками:

    ```
    <input type="password">
    ```

  - `file` — поле ввода имени файла с кнопкой **Обзор**. Позволяет отправить файл на Web-сервер:

    ```
    <input type="file">
    ```

  - `checkbox` — поле для установки флажка, который можно установить или сбросить:

    ```
    <input type="checkbox">
    ```

  - `radio` — элемент-переключатель (иногда их называют *радиокнопками*):

    ```
    <input type="radio">
    ```

  - `reset` — кнопка, при нажатии которой вся форма очищается. Точнее сказать, все элементы формы получают значения по умолчанию:

    ```
    <input type="reset" value="Очистить">
    ```

  - `submit` — кнопка, при нажатии которой происходит отправка данных, введенных в форму:

    ```
    <input type="submit" value="Отправить">
    ```

  - `button` — обычная командная кнопка:

    ```
    <input type="button" value="OK"
           onclick="alert('OK')">
    ```

    Такую кнопку имеет смысл использовать только с прикрепленным к ней скриптом. Как это сделать, будет показано в *главе 3*;

  - `image` — изображение, при щелчке на котором выполняется отправка данных формы. При этом отправляются также и координаты щелчка мыши на изображении. Путь к файлу с изображением указывается в параметре `src`, а в параметре `alt` задается альтернативный текст:

    ```
    <input type="image" src="button.gif" alt="Отправить">
    ```

  - `hidden` — скрытый элемент, значение которого отправляется вместе со всеми данными формы. Элемент не показывается пользователю, но позволяет хра-

нить, например, данные из предыдущей формы (если пользователь последовательно заполняет несколько форм) или уникальное имя пользователя:

```
<input type="hidden" name="user_id" value="1">
```

- `name` — задает имя элемента управления. Оно должно обязательно указываться латинскими буквами (например, `field`) или комбинацией латинских букв и цифр (например, `field1`). Имя элемента не должно начинаться с цифры:

```
<input type="text" name="field1">
```

- `disabled` — запрещает доступ к элементу формы. При наличии параметра элемент отображается серым цветом:

```
<input type="text" name="field2" disabled>
```

- `readonly` — указывает, что элемент доступен только для чтения. При наличии параметра значение элемента изменить нельзя:

```
<input type="text"
      value="Значение только для чтения" readonly>
```

- `value` — задает значение по умолчанию для элементов и текст надписи для кнопок:

```
<input type="text" value="Значение по умолчанию">
<input type="submit" value="Отправить">
```

В HTML 5 добавлены следующие элементы (значения параметра `type`):

- `url` — поле ввода URL-адреса. Значение автоматически проверяется. Форма отправляется только в случае, если поле не заполнено или содержит корректное значение URL-адреса. Существование URL-адреса не проверяется. Если нужно исключить пустое значение, то следует дополнительно указать параметр `required`:

```
<input type="url" required>
```

- `email` — поле ввода адреса электронной почты. Значение автоматически проверяется. Форма отправляется только в случае, если поле не заполнено или содержит корректное значение E-mail. Существование E-mail не проверяется. Если нужно исключить пустое значение, то следует дополнительно указать параметр `required`:

```
<input type="email" required>
```

- `tel` — поле для ввода телефона. Значение автоматически не проверяется, т. к. форматы номеров сильно различаются. Мобильные Web-браузеры для этого поля могут отобразить специальную клавиатуру. В остальных Web-браузерах элемент выглядит как обычное текстовое поле:

```
<input type="tel" placeholder="+0 (000) 000-00-00">
```

- `number` — поле для ввода числа. Справа от поля Web-браузер отображает две кнопки, с помощью которых можно увеличить или уменьшить значение на шаг,

указанный в параметре step. Мобильные Web-браузеры для этого поля отображают цифровую клавиатуру. Значение автоматически проверяется. Форма отправляется только в случае, если поле не заполнено или содержит положительное или отрицательное число. Если нужно исключить пустое значение, то следует дополнительно указать параметр required:

```
<input type="number" step="5" required>
<input type="number" step="0.5" required>
```

- range — ползунок, с помощью которого можно выбрать числовое значение из диапазона. К сожалению, элемент не содержит маркеров и меток, поэтому выставить значение можно только примерно. Для информирования пользователя о точном текущем значении нужно использовать дополнительные скрипты. Минимальное значение задается параметром min, максимальное — параметром max, текущее — параметром value, а шаг — параметром step:

```
<input type="range" min="0" max="100" value="10">
```

- color — инструмент для выбора цвета. При щелчке на элементе отображается диалоговое окно (рис. 1.5), в котором можно выбрать цвет или ввести его значение вручную. Элемент не поддерживается браузером Internet Explorer. Текущее значение задается с помощью параметра value в формате #RRGGBB:

```
<input type="color" value="#FF0000">
```

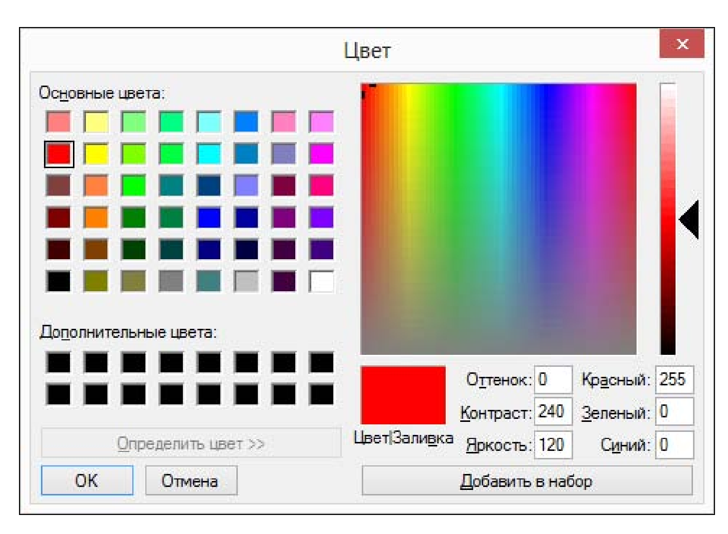


Рис. 1.5. Диалоговое окно для выбора цвета

- search — поле ввода подстроки для поиска:

```
<input type="search">
```

- date — поле для ввода даты (элемент не поддерживается браузерами Firefox и Internet Explorer):

```
<input type="date">
```

- `time` — поле для ввода времени (элемент не поддерживается браузерами Firefox и Internet Explorer):

  ```
  <input type="time">
  ```

- `datetime-local` — поле для ввода локальной даты и времени (элемент не поддерживается браузерами Firefox и Internet Explorer):

  ```
  <input type="datetime-local">
  ```

- `month` — поле для ввода месяца и года (элемент не поддерживается браузерами Firefox и Internet Explorer):

  ```
  <input type="month">
  ```

- `week` — поле для ввода номера недели и года (элемент не поддерживается браузерами Firefox и Internet Explorer):

  ```
  <input type="week">
  ```

В HTML 5 тег `<input>` получил новые параметры:

- `autocomplete` — аналогичен одноименному параметру тега `<form>`. Значение `on` включает автодополнение, `off` — отключает:

  ```
  <input type="text" autocomplete="off">
  ```

- `autofocus` — устанавливает фокус ввода на элемент:

  ```
  <input type="text" name="field2" autofocus>
  ```

- `placeholder` — задает текст подсказки, который будет выводиться прямо в поле ввода, пока поле не заполнено:

  ```
  <input type="text" placeholder="Введите имя">
  ```

- `required` — если параметр указан, то поле обязательно для заполнения:

  ```
  <input type="text" required>
  ```

- `pattern` — шаблон вводимого значения в виде регулярного выражения JavaScript:

  ```
  <input type="tel" placeholder="+0 (000) 000-00-00" required
  pattern="\+[1-9] \([0-9]{3}\) [0-9]{3}-[0-9]{2}-[0-9]{2}">
  ```

- `form` — связывает элемент с формой по ее идентификатору (применяется только, если элемент находится вне формы):

  ```
  <form action="process.php" id="form1"></form>
  <input type="text" name="field1" form="form1">
  ```

Новые элементы сначала проверяют, является ли введенное значение корректным числом, адресом электронной почты, URL-адресом, датой, временем и т. д., и только в случае успешного выполнения этой проверки данные формы отправляются на сервер. Если проверка не увенчается успехом, данные отправлены не будут, и появится сообщение о некорректном вводе данных. Мы можем отключить эту про-

верку, указав в теге `<form>` параметр `novalidate`. Однако делать это следует лишь в особых случаях.

Если пользователь не ввел значение в элемент управления, помеченный как обязательный к заполнению (указан параметр `required`), или если это значение не соответствует заданным параметрам (не укладывается в указанный диапазон или не совпадает с шаблоном регулярного выражения из параметра `pattern`), то появится соответствующее предупреждение. Понятно, что данные на сервер в этом случае отправлены не будут.

> ***Примечание***
>
> Следует учитывать, что злоумышленник может изменить HTML-код, поэтому на Web-сервере нужно в обязательном порядке выполнить проверку полученных данных, а не полагаться на Web-браузер. Тем не менее, проверка на стороне клиента позволит снизить нагрузку на Web-сервер, поэтому отказываться от такой проверки не следует. Данные нужно проверять и в Web-браузере, и на Web-сервере.

Остальные параметры специфичны для каждого отдельного элемента. Поэтому рассмотрим каждый тип элемента отдельно.

Текстовое поле и поле ввода пароля

Для текстового поля и поля ввода пароля используются следующие параметры:

- `value` — задает текст поля по умолчанию:

  ```
  <input type="text" name="field1" value="http://">
  ```

- `maxlength` — указывает максимальное количество символов, которое может быть введено в поле:

  ```
  <input type="text" name="field1" maxlength="100">
  ```

- `size` — определяет видимый размер поля ввода:

  ```
  <input type="text" name="field1" maxlength="100" size="20">
  ```

В HTML 5 доступен также параметр `minlength`, задающий минимальное количество символов (параметр не поддерживается браузером Internet Explorer):

```
<input type="text" name="field1" minlength="3">
```

Кнопки *Сброс*, *Отправить* и командная кнопка

Для этих кнопок указывается один параметр: `value` — он задает текст, отображаемый на кнопке:

```
<input type="submit" value="Отправить">
```

Скрытое поле *hidden*

Для скрытого поля указывается один параметр: `value` — он определяет значение скрытого поля:

```
<input type="hidden" name="hidden1" value="1">
```

Поле для установки флажка

Для полей-флажков используются следующие параметры:

- `value` — задает значение, которое будет передано Web-серверу, если флажок отмечен. Если флажок снят, значение не передается. Если параметр не задан, используется значение по умолчанию — `on`:

  ```
  <input type="checkbox" name="check1" value="yes">Текст
  ```

- `checked` — указывает, что флажок по умолчанию установлен:

  ```
  <input type="checkbox" name="check1" value="yes" checked>Текст
  ```

Элементы `checkbox` можно объединить в группу. Для этого необходимо установить одинаковое значение параметра `name`. Чтобы получить все значения на сервере, после названия поля следует указать квадратные скобки (это признак массива в языке PHP):

```
<input type="checkbox" name="check[]" value="1">Текст 1
<input type="checkbox" name="check[]" value="2">Текст 2
<input type="checkbox" name="check[]" value="3">Текст 3
```

Элемент-переключатель

При описании элемента-переключателя используются такие параметры:

- `value` — указывает значение, которое будет передано Web-серверу, если переключатель выбран:

  ```
  <input type="radio" name="radio1" value="yes">Текст
  ```

  Если ни одно из значений не выбрано, никаких данных передано не будет;

- `checked` — обозначает переключатель, выбранный по умолчанию:

  ```
  <input type="radio" name="radio1" value="yes" checked>Текст
  ```

Элемент-переключатель может существовать только в составе группы подобных элементов, из которых может быть выбран только один. Для объединения переключателей в группу необходимо установить одинаковое значение параметра `name` и разное значение параметра `value`:

```
Укажите ваш пол:<br>
<input type="radio" name="sex" value="male" checked>Мужской
<input type="radio" name="sex" value="female">Женский
```

Поле выбора файла

Поле для выбора файла поддерживает следующие параметры:

- `multiple` — если параметр указан, то можно выбрать сразу несколько файлов:

  ```
  <input type="file" multiple>
  ```

- accept — задает поддерживаемые MIME-типы или расширения файлов (значения приводятся через запятую):

```
<input type="file" accept="image/jpeg,image/png,image/gif">
<input type="file" accept="image/*">
<input type="file" accept=".gif,.jpg,.jpeg">
```

Элементы для ввода числа и выбора значения из диапазона

Элементы number и range поддерживают следующие параметры:

- value — текущее значение;
- min — минимальное значение;
- max — максимальное значение;
- step — шаг.

Пример:

```
<input type="range" min="0" max="100" value="10" step="2">
```

Элемент для ввода даты

Элемент date поддерживает следующие параметры:

- value — текущее значение;
- min — минимальное значение;
- max — максимальное значение.

Значение указывается в формате ГГГГ-ММ-ДД:

```
<input type="date" min="2010-01-01" max="2020-01-01" value="2018-01-01">
```

1.11.5. Список автодополнения

Автодополнение значений, заносимых в поле ввода, — очень полезная вещь. Впрочем, не всегда, и именно поэтому HTML 5 позволяет нам отключить его, указав для параметра autocomplete значение off. Но мы можем сделать его еще лучше, задав для какого-либо поля ввода несколько предопределенных значений, которые будут подставлены в список автодополнения.

В HTML 5 этот список создается с помощью парного тега <datalist> (он не поддерживается браузером Safari). Для него мы в обязательном порядке с помощью параметра id укажем идентификатор.

Позиции для списка автодополнения создаются с помощью тегов <option>. Эти теги помещаются внутрь тега <datalist>.

И напоследок мы привяжем созданный список к полю ввода, задав в его теге идентификатор списка в качестве значения параметра list (листинг 1.20). Результат показан на рис. 1.6.

Листинг 1.20. Пример автодополнения

```
<datalist id="browsers">
  <option value="Internet Explorer"></option>
  <option value="Firefox"></option>
  <option value="Chrome"></option>
  <option value="Opera"></option>
  <option value="Safari"></option>
</datalist>
<input type="text" name="browser" list="browsers"
       placeholder="Ваш Web-браузер">
```

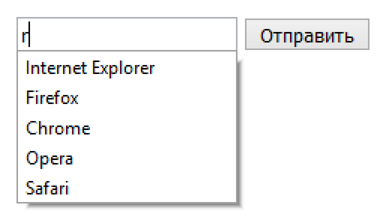


Рис. 1.6. Список автодополнения

1.11.6. Тег <*textarea*>. Текстовая область

Парный тег `<textarea>` создает внутри формы поле для ввода многострочного текста:

```
<textarea>
Текст по умолчанию
</textarea>
```

В окне Web-браузера поле отображается в виде прямоугольной области с полосами прокрутки.

Тег имеет следующие параметры:

- `name` — уникальное имя поля:

  ```
  <textarea name="txt">
  Текст по умолчанию
  </textarea>
  ```

- `cols` — ширина поля:

  ```
  <textarea name="txt" cols="15">
  Текст по умолчанию
  </textarea>
  ```

- `rows` — число строк видимого текста:

  ```
  <textarea name="txt" cols="15" rows="10">
  Текст по умолчанию
  </textarea>
  ```

- disabled — запрещает доступ к элементу формы. При наличии параметра элемент отображается серым цветом:

```
<textarea name="txt" disabled></textarea>
```

- readonly — указывает, что элемент доступен только для чтения. При наличии параметра значение элемента изменить нельзя:

```
<textarea name="txt" readonly>
Значение только для чтения
</textarea>
```

В HTML 5 тег `<textarea>` получил новые параметры:

- autocomplete — аналогичен одноименному параметру тега `<form>`. Значение on включает автодополнение, off — отключает:

```
<textarea name="txt" autocomplete="off">
Текст по умолчанию
</textarea>
```

- autofocus — устанавливает фокус ввода на элемент:

```
<textarea name="txt" autofocus></textarea>
```

- placeholder — задает текст подсказки, который будет выводиться прямо в поле ввода, пока поле не заполнено:

```
<textarea name="txt" placeholder="Введите текст"></textarea>
```

- required — если параметр указан, то поле обязательно для заполнения:

```
<textarea name="txt" required></textarea>
```

- maxlength — указывает максимальное количество символов, которое может быть введено в поле:

```
<textarea name="txt" maxlength="100"></textarea>
```

- minlength — задает минимальное количество символов (параметр не поддерживается браузером Internet Explorer):

```
<textarea name="txt" minlength="3"></textarea>
```

- wrap — управляет автоматической вставкой символов переноса строк в текст, если он не помещается на строке. Может принимать следующие значения:
  - soft — слова автоматически переносятся на новую строку, но на сервер символы переноса строк не отправляются (значение по умолчанию);
  - hard — текст переносится на новую строку и символы переноса отправляются на сервер (в этом случае требуется указать параметр cols):

```
<textarea name="txt1" wrap="soft"></textarea>
<textarea name="txt2" wrap="hard" cols="20"></textarea>
```

- form — связывает элемент с формой по ее идентификатору (применяется только, если элемент находится вне формы):

```
<form action="process.php" id="form1"></form>
<textarea name="txt1" form="form1"></textarea>
```

В CSS 3 атрибут стиля `resize` позволяет указать, можно ли изменять размеры текстовой области с помощью мыши. Он может принимать следующие значения:

- `none` — размеры поля изменить нельзя;
- `horizontal` — можно только по горизонтали;
- `vertical` — можно только по вертикали;
- `both` — размеры поля можно менять и по горизонтали, и по вертикали.

Пример указания атрибута `resize` в параметре `style`:

```
<textarea cols="25" rows="10" style="resize: both">
Текст по умолчанию
</textarea>
```

1.11.7. Тег *<select>*. Список с предопределенными значениями

Тег `<select>` создает внутри формы список с возможными значениями:

```
<select>
<option>Элемент1</option>
<option>Элемент2</option>
</select>
```

Тег `<select>` имеет следующие параметры:

- `name` — задает уникальное имя списка:

  ```
  <select name="select1">
  ```

- `size` — определяет число одновременно видимых элементов списка:

  ```
  <select name="select1" size="3">
  ```

  По умолчанию `size` имеет значение `1`;

- `multiple` — указывает, что из списка можно выбрать сразу несколько элементов одновременно. Чтобы получить все значения на сервере, после названия списка следует указать квадратные скобки (это признак массива в языке PHP):

  ```
  <select name="select[]" size="3" multiple>
  ```

- `disabled` — запрещает доступ к элементу формы. При наличии параметра элемент отображается серым цветом:

  ```
  <select name="select1" disabled>
  ```

В HTML 5 тег `<select>` получил новые параметры:

- `autofocus` — устанавливает фокус ввода на элемент:

  ```
  <select name="select1" autofocus>
  ```

- `required` — если параметр указан, то в списке обязательно должен быть выбран пункт:

```
<select name="select1" size="2" required>
<option>Элемент1</option>
<option>Элемент2</option>
</select>
```

- `form` — связывает элемент с формой по ее идентификатору (применяется только, если элемент находится вне формы):

```
<form action="process.php" id="form1"></form>
<select name="select1" form="form1">
<option>Элемент1</option>
<option>Элемент2</option>
</select>
```

Внутри тегов `<select>` и `</select>` располагаются теги `<option>`, с помощью которых описывается каждый элемент списка. Закрывающий тег не обязателен.

Тег `<option>` имеет следующие параметры:

- `value` — задает значение, которое будет передано Web-серверу, если пункт списка выбран. Если параметр не задан, посылается текст этого пункта:

```
<select name="select1">
<option value="val1">Элемент1</option>
<option>Элемент2</option>
</select>
```

  Если выбран пункт `Элемент1`, то посылается:

```
select1=val1
```

  Если выбран пункт `Элемент2`, то посылается:

```
select1=Элемент2
```

- `selected` — указывает, какой пункт списка выбран изначально:

```
<select name="select1">
<option value="val1">Элемент1</option>
<option selected>Элемент2</option>
</select>
```

- `disabled` — делает пункт списка недоступным. При наличии параметра пункт отображается серым цветом:

```
<select name="select1">
<option value="val1">Элемент1</option>
<option disabled>Элемент2</option>
</select>
```

С помощью тега `<optgroup>` можно объединить несколько пунктов в группу. Название группы указывается в параметре `label`:

```
<select name="select1">
  <optgroup label="Отечественные">
    <option value="1">ВАЗ</option>
    <option value="2">ГАЗ</option>
    <option value="3">Москвич</option>
  </optgroup>
  <optgroup label="Зарубежные">
    <option value="4">BMW</option>
    <option value="5">Opel</option>
    <option value="6">Audi</option>
  </optgroup>
</select>
```

Результат показан на рис. 1.7.

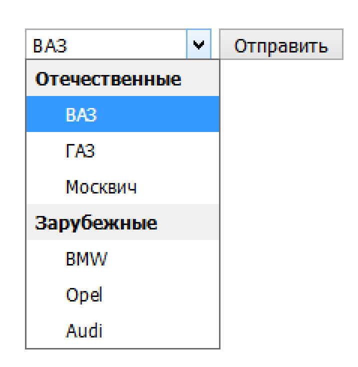


Рис. 1.7. Объединение нескольких пунктов списка в группу

Если в теге <optgroup> указать параметр disabled, то вся группа станет недоступной:

```
<optgroup label="Зарубежные" disabled>
```

1.11.8. Тег <*button*>. Кнопка

Парный тег <button> позволяет вставить кнопку. Внутри тегов <button> и </button> могут находиться различные элементы или просто текст, задающие содержимое, отображаемое на кнопке. Пример вывода значка и текста:

```
<img src="icon.gif" alt="" width="16" height="16"
     style="vertical-align: middle"> Отправить</button>
```

Тег <button> имеет следующие параметры:

- type — задает тип кнопки:
  - submit — кнопка, при нажатии которой происходит отправка данных, введенных в форму (значение по умолчанию):

```
<button type="submit">Отправить</button>
```

* reset — кнопка, при нажатии которой вся форма очищается. Точнее сказать, все элементы формы получают значения по умолчанию:

  ```
  <button type="reset">Очистить</button>
  ```

* button — обычная командная кнопка:

  ```
  <button type="button" onclick="alert('OK')">OK</button>
  ```

- name — задает имя кнопки;
- value — задает значение, отправляемое на сервер (name=value):

  ```
  <button name="btn1" value="OK">Отправить</button>
  ```

- disabled — запрещает доступ к кнопке. При наличии параметра кнопка отображается серым цветом:

  ```
  <button disabled>Отправить</button>
  ```

В HTML 5 тег <button> получил новые параметры:

- autofocus — устанавливает фокус ввода на кнопку:

  ```
  <button autofocus>Отправить</button>
  ```

- form — связывает кнопку с формой по ее идентификатору (применяется только, если элемент находится вне формы);
- formaction — URL-адрес обработчика формы;
- formenctype — задает способ кодирования передаваемых данных. Может принимать значения: application/x-www-form-urlencoded (применяется по умолчанию) или multipart/form-data (указывается при пересылке Web-серверу файлов);
- formmethod — определяет, как будут пересылаться данные из формы Web-серверу. Может принимать два значения: GET и POST;
- formtarget — указывает, куда будет помещен документ, являющийся результатом обработки данных формы Web-сервером;
- formnovalidate — если присутствует, то значения, занесенные в поля ввода, не будут проверяться:

  ```
  <form action="process.php" id="form1"></form>
  <button form="form1" formaction="test.php" formmethod="GET"
          formenctype="application/x-www-form-urlencoded"
          formtarget="_blank" formnovalidate="formnovalidate">
          Отправить</button>
  ```

1.11.9. Тег *<label>*

С помощью тега <label> можно указать пояснительную надпись для элемента формы. Тег имеет два параметра:

- for — позволяет указать идентификатор элемента, к которому привязана надпись. Точно такой же идентификатор должен быть указан в параметре id элемента формы:

```
<label for="passw1">Пароль*:</label>
<input type="password" name="passw1" id="passw1">
```

Если элемент формы разместить внутри тега `<label>`, то надпись будет связана с элементом и без указания параметра `for`:

```
<label>Пароль*:
  <input type="password" name="passw1" id="passw1">
</label>
```

- `accesskey` — задает клавишу быстрого доступа. При нажатии этой клавиши одновременно с клавишей <Alt> (в браузерах Internet Explorer, Chrome и Safari), <Shift>+<Alt> (в браузере Firefox) или <Shift>+<Esc> (в браузере Opera) элемент окажется в фокусе ввода:

```
<label accesskey="N">Пароль*:
  <input type="password" name="passw1" id="passw1">
</label>
```

В качестве примера рассмотрим форму регистрации пользователя (листинг 1.21), а заодно продемонстрируем использование CSS для форматирования страницы.

Листинг 1.21. Пример формы регистрации пользователя

```
<!DOCTYPE html>
<html lang="ru">
<head>
 <meta charset="utf-8">
 <title>Пример формы регистрации пользователя</title>
 <style type="text/css">
   body { /* Стиль для всего документа */
     font-size: 10pt; /* Размер шрифта */
     font-family: "Verdana", sans-serif; /* Название шрифта */
   }
   label { /* Стиль для всех элементов label */
     display: inline-block; /* Тип блока */
     width: 150px; /* Ширина */
     vertical-align: top; /* Вертикальное выравнивание */
   }
   select { /* Стиль для всех списков */
     width: 250px; /* Ширина */
     border: 1px solid black; /* Определение стиля для границы */
   }
   input.txt { /* Стиль для элемента input, имеющего класс txt */
     width: 250px; /* Ширина */
     border: 1px solid black; /* Определение стиля для границы */
   }
   textarea { /* Стиль для многострочного текстового поля */
     width: 250px; /* Ширина */
```

```
      height: 100px; /* Высота */
      border: 1px solid black; /* Определение стиля для границы */
   }
   form div { /* Стиль для всех div, расположенных внутри form */
      margin-bottom: 20px; /* Отступ блока снизу */
   }
 </style>
</head>
<body>
 <h1>Пример формы регистрации пользователя</h1>
 <form action="reg.php" method="POST" enctype="multipart/form-data">
  <div><label for="login">Логин*:</label>
   <input type="text" name="login" id="login" class="txt" required>
  </div>
  <div><label for="passw1">Пароль*:</label>
   <input type="password" name="passw1" id="passw1"
          class="txt" required>
  </div>
  <div><label for="passw2">Повторите пароль*:</label>
   <input type="password" name="passw2" id="passw2"
          class="txt" required>
  </div>
  <div><label for="sex1">Пол*:</label>
   Муж. <input type="radio" name="sex" id="sex1" value="1">
   Жен. <input type="radio" name="sex" id="sex2" value="2"></div>
  <div><label for="education">Образование*:</label>
   <select id="education" name="education" required>
    <option value="" disabled> </option>
    <option value="1">Среднее</option>
    <option value="2">Высшее</option>
   </select></div>
  <div><label for="comment">Комментарий:</label>
   <textarea id="comment" name="comment" cols="15" rows="10"></textarea>
  </div>
  <div><label for="userfile">Ваше фото:</label>
   <input type="file" name="userfile" id="userfile"></div>
  <div><label for="rule">С правилами согласен*:</label>
   <input type="checkbox" name="rule" id="rule" value="yes" required>
  </div>
  <div>
    <input type="submit" value="Отправить" style="margin-left: 150px;">
  </div>
 </form>
</body>
</html>
```

1.11.10. Группировка элементов формы

Парный тег <fieldset> позволяет сгруппировать элементы формы. Web-браузеры вокруг группы отображают рамку. На линии рамки с помощью тега <legend> можно разместить надпись:

```
<fieldset>
   <legend>Пол</legend>
   Муж. <input type="radio" name="sex" value="1">
   Жен. <input type="radio" name="sex" value="2">
</fieldset>
```

1.12. Тег <*meter*>

С помощью парного тега <meter> в HTML 5 можно отобразить значение из диапазона в графической форме (не поддерживается браузером Internet Explorer). Тег имеет следующие параметры:

- value — текущее значение;
- min — минимальное значение;
- max — максимальное значение;
- optimum — оптимальное значение;
- low — низкое значение;
- high — высокое значение.

Пример:

```
<meter value="10" min="0" max="100" optimum="50" low="25" high="75">
Web-браузер не поддерживает элемент meter
</meter>
<meter value="50" min="0" max="100" optimum="50" low="25" high="75">
</meter>
<meter value="80" min="0" max="100" optimum="50" low="25" high="75">
</meter>
```

1.13. Тег <*progress*>. Индикатор хода процесса

В HTML 5 доступен парный тег <progress>, позволяющий в графической форме отобразить текущее состояние хода длительного процесса. Тег имеет следующие параметры:

- value — текущее значение;
- max — максимальное значение.

Пример:

```
<progress max="100" value="75"></progress>
```

Обычно тег имеет смысл использовать совместно со скриптом на языке JavaScript.

1.14. Аудио и видео

В HTML 5 добавлена поддержка вставки на Web-страницы аудио- и видеороликов без применения сторонних программ, но, к сожалению, разные Web-браузеры поддерживают различные форматы аудио и видео.

1.14.1. Вставка аудиоролика

Аудиоролик на Web-страницу вставляется с помощью тега <audio>:

```
<audio src="sound.mp3" controls preload="metadata">
<p>Ваш Web-браузер не поддерживает элемент audio</p>
</audio>
```

Если Web-браузер не поддерживает тег <audio>, то будет выведено соответствующее сообщение, расположенное между тегами <audio> и </audio>.

В Интернете используются следующие основные форматы аудио: MP3, AAC, OGG/Vorbis и WAV. Первые два формата поддерживаются всеми основными современными Web-браузерами. Формат OGG/Vorbis не поддерживается браузерами Internet Explorer и Safari. Формат WAV не поддерживается браузером Internet Explorer.

Тег <audio> содержит следующие параметры:

- src — путь к файлу с аудиозаписью;
- autoplay — если указан, воспроизведение ролика начнется сразу после загрузки страницы:

  ```
  <audio src="sound.mp3" autoplay></audio>
  ```

- controls — если параметр указан, на экране будут присутствовать элементы для управления воспроизведением ролика (шкала процесса с регулятором позиции воспроизведения, регулятор громкости, кнопки пуска и паузы и кнопка отключения звука);
- loop — если указан, ролик будет воспроизводиться бесконечно («зациклится»):

  ```
  <audio src="sound.mp3" autoplay loop></audio>
  ```

- muted — если указан, после загрузки страницы звук будет отключен;
- preload — позволяет указать, как будет выполнена предварительная загрузка файла с роликом. Доступны три значения: none (не выполнять предварительную загрузку), metadata (загрузить только самое начало файла, где хранятся сведения о ролике, в частности, его продолжительность) и auto (загрузить весь файл, поведение по умолчанию). Этот параметр имеет смысл указывать лишь в том случае, если параметр autoplay отсутствует в теге.

1.14.2. Вставка видеоролика

Для вставки на Web-страницу видеоролика применяется тег `<video>`:

```
<video src="video.mp4" controls width="500">
<p>Ваш Web-браузер не поддерживает элемент video</p>
</video>
```

Если Web-браузер не поддерживает тег `<video>`, то будет выведено соответствующее сообщение, расположенное между тегами `<video>` и `</video>`.

В Интернете используются следующие основные форматы видео: MPEG 4/H.264, OGG/Theora и WebM. Первый формат поддерживается всеми основными современными Web-браузерами. Два последних не поддерживаются браузерами Internet Explorer и Safari.

Тег `<video>` содержит следующие параметры:

- `src` — путь к файлу с видеозаписью;
- `autoplay` — если указан, воспроизведение ролика начнется сразу после загрузки страницы;
- `controls` — если параметр указан, на экране будут присутствовать элементы для управления воспроизведением ролика (шкала процесса с регулятором позиции воспроизведения, регулятор громкости, кнопки пуска и паузы, кнопка отключения звука и кнопка полноэкранного режима);
- `loop` — если указан, ролик будет воспроизводиться бесконечно («зациклится»):

  ```
  <video src="video1.mp4" controls autoplay loop></video>
  ```

- `muted` — если указан, после загрузки страницы звук будет отключен;
- `preload` — позволяет указать, как будет выполнена предварительная загрузка файла с роликом. Доступны три значения: `none` (не выполнять предварительную загрузку), `metadata` (загрузить только самое начало файла, где хранятся сведения о ролике, в частности, его продолжительность) и `auto` (загрузить весь файл, поведение по умолчанию). Этот параметр имеет смысл указывать лишь в том случае, если параметр `autoplay` отсутствует в теге;
- `width` — ширина области воспроизведения (в пикселах);
- `height` — высота области воспроизведения (в пикселах). Если ширина и высота не указаны, то будут использоваться размеры видео. Если указан только один параметр, то значение второго будет рассчитано автоматически с соблюдением пропорций;
- `poster` — путь к изображению, которое будет отображаться в области воспроизведения. Если параметр не указан, будет выведен первый кадр ролика:

  ```
  <video src="video.mp4" controls width="640" height="480"
         poster="poster.jpg"></video>
  ```

1.14.3. Указание нескольких источников аудио или видео

Ранее говорилось, что разные Web-браузеры поддерживают различные форматы аудио- и видеофайлов. В связи с этим возникает вопрос: можно ли указать в теге <audio> или <video> несколько вариантов, чтобы Web-браузер загрузил тот, который он поддерживает?

Можно. Требуется лишь удалить из тега <audio> или <video> параметр src и поместить в сам этот тег набор тегов <source>, каждый из которых укажет один из вариантов помещаемого на страницу файла:

```
<video controls>
  <source src="video.mp4">
  <source src="video.webm">
</video>
```

Здесь мы указываем в теге <video> два ролика (в форматах MP4 и WebM).

Тег <source> содержит следующие параметры:

- src — путь к файлу с роликом;
- type — MIME-тип файла с роликом: video/mp4 (для MPEG 4), video/webm (для WebM), audio/mpeg (для MP3), audio/ogg (для OGG) и др. Если параметр не указан, Web-браузер сам определит формат файла.

1.14.4. Тег *<track>*

Тег <track> позволяет добавить к мультимедиа текстовую дорожку — например, субтитры к видео. Тег имеет следующие параметры:

- src — задает путь к файлу с текстовой дорожкой;
- srclang — указывает код языка;
- kind — задает тип дорожки: subtitles (субтитры), captions (заголовки), descriptions (описания), chapters (главы) и metadata (метаданные);
- label — название дорожки;
- default — если параметр указан, то определяет дорожку по умолчанию.

Вот пример добавления русских и английских субтитров к видео:

```
<video controls width="500">
  <source src="video.mp4">
  <source src="video.webm">
  <track src="subtitles.ru.vtt" srclang="ru" kind="subtitles"
         label="Russian" default>
  <track src="subtitles.en.vtt" srclang="en" kind="subtitles"
         label="English">
  <p>Ваш Web-браузер не поддерживает элемент video</p>
</video>
```

Содержимое файла subtitles.ru.vtt:

```
WEBVTT FILE

00:00.000 --> 00:01.000
Эпизод 1

00:02.000 --> 00:20.000
Эпизод 2
```

1.15. Универсальные параметры

В предыдущих разделах мы рассмотрели параметры, характерные для конкретных тегов. Однако существуют также универсальные параметры, которые содержат большинство тегов. Давайте рассмотрим наиболее важные универсальные параметры:

- `id` — задает уникальный идентификатор элемента. С помощью этого идентификатора мы можем обратиться к элементу из CSS или из скрипта на языке JavaScript. Имя идентификатора должно начинаться с буквы и может содержать латинские буквы, цифры и символ подчеркивания:

  ```
  <div id="mydiv"></div>
  ```

  Пример обращения из CSS (задаем цвет фона и высоту элемента):

  ```
  <style type="text/css">
  #mydiv {
     background: green;
     height: 100px
  }
  </style>
  ```

  Пример обращения из скрипта на JavaScript (вставляем текст):

  ```
  <script>
  var mydiv = document.getElementById('mydiv');
  mydiv.innerHTML = 'Текст внутри элемента с id mydiv';
  </script>
  ```

- `class` — задает стилевой класс для элемента:

  ```
  <div class="mydiv"></div>
  ```

  С помощью этого класса мы можем обратиться к элементу из CSS (обратите внимание: перед именем класса мы указываем точку, а не символ #):

  ```
  <style type="text/css">
  .mydiv {
     background: green;
     height: 100px
  }
  </style>
  ```

- `style` — позволяет указать стилевое оформление для элемента:

```
<div style="background: green; height: 100px"></div>
```

- `title` — задает текст всплывающей подсказки, отображаемой при наведении указателя мыши на элемент:

```
<div style="background: green; height: 100px"
     title="Текст подсказки">Текст</div>
```

- `hidden` — скрывает элемент от просмотра (отобразить элемент можно с помощью скрипта на языке JavaScript):

```
<div style="background: green; height: 100px" hidden>
Текст</div>
```

- `tabindex` — задает порядок обхода элементов при нажатии клавиши <Tab>:

```
<input type="text" tabindex="2">
<input type="text" tabindex="1">
```

- `data-<Имя>` — позволяет создать пользовательский параметр с произвольным именем (параметр доступен в HTML 5):

```
<div style="background: green; height: 100px" data-mydata="5">
Текст</div>
```

- `on*` — группа параметров, позволяющих назначить обработчик какого-либо события:

```
<button type="button" onclick="alert('OK')">OK</button>
```

Эти параметры мы рассмотрим при изучении языка JavaScript.

1.16. Проверка HTML-документов на соответствие стандартам

После создания HTML-документа его необходимо проверить на отсутствие ошибок и соответствие стандартам. Ведь можно случайно забыть закрыть тег, допустить опечатку в названии тега или параметра, нарушить правильность вложенности тегов и др. Web-браузеры обычно не сообщают об ошибках, а пытаются их обработать. Поэтому о существовании ошибок можно узнать только в случае, если Web-браузер неправильно их обработал, и это видно в его окне.

Для проверки HTML-документов предназначены сайты **http://validator.w3.org/** и **https://html5.validator.nu/**. Чтобы проверить документ, размещенный в Интернете, достаточно ввести URL-адрес и нажать кнопку **Check**. Можно также загрузить файл или вставить HTML-код в поле ввода многострочного текста. Если после проверки были обнаружены ошибки, то будет выведено их подробное описание. После исправления ошибок следует повторно проверить HTML-документ.



ГЛАВА 2

Основы CSS 3. Форматируем Web-страницу с помощью стилей

2.1. Способы встраивания определения стиля

Каскадные таблицы стилей (*Cascading Style Sheets, CSS*) позволяют существенно расширить возможности языка HTML за счет более гибкого управления форматированием Web-страницы. Применение стилей позволяет задавать точные характеристики практически всех элементов Web-страницы, а это значит, что можно точно контролировать внешний вид Web-страницы в окне Web-браузера.

Рассмотрим пример указания размера шрифта для абзаца:

```
<p style="font-size: 12pt">Текст</p>
```

и, прежде чем приступить к изучению CSS, разберемся с основными понятиями.

Значение параметра `style` (`font-size: 12pt`) называется *определением стиля* или *стилем*. Элемент определения стиля (`font-size`) называется *атрибутом*. Каждый атрибут имеет *значение* (`12pt`), указываемое после двоеточия. Совокупность определений стилей, вынесенных в заголовок HTML-документа или в отдельный файл, называют *таблицей стилей*.

В этом издании книги мы будем изучать третью версию CSS. Как и в HTML 5, поддержка атрибутов CSS 3 зависит от Web-браузеров. Если при описании атрибута не указана версия CSS, то эти возможности были доступны в CSS 2. Для новых атрибутов, появившихся в CSS 3, мы будем явно указывать версию. Поэтому, если есть большое желание поддерживать старые версии Web-браузеров, то атрибуты, обозначенные версией CSS 3, лучше не использовать.

> ***Примечание***
>
> Получить полную информацию о текущей поддержке атрибутов стиля Web-браузерами можно на сайте **https://caniuse.com/**.

Задать стиль можно тремя способами: встроить определение стиля в тег, встроить определения стилей в заголовок HTML-документа или вынести таблицу стилей в отдельный файл.

2.1.1. Встраивание определения стиля в тег

Определение стиля, как и было отмечено ранее, встраивается в любой тег с помощью параметра `style`:

```
<p style="font-size: 12pt">Текст</p>
```

Обратите внимание: параметр `style` поддерживают все теги.

Если определение стиля состоит из нескольких атрибутов, то они указываются через точку с запятой:

```
<p style="font-size: 12pt; color: red">Текст</p>
```

Если какое-либо значение атрибута требует наличия кавычек, то оно указывается в апострофах:

```
<p style="font-size: 12pt; color: red; font-family: 'Times New Roman'">
Текст</p>
```

2.1.2. Встраивание определения стилей в заголовок HTML-документа

Все определения стилей можно собрать в одном месте (листинг 2.1). В этом случае стили указываются между тегами `<style>` и `</style>`. Сам тег `<style>` должен быть расположен в разделе `HEAD` HTML-документа.

Листинг 2.1. Пример использования стилей

```
<!DOCTYPE html>
<html lang="ru">
<head>
 <meta charset="utf-8">
 <title>Пример использования стилей</title>
 <style type="text/css">
  body {                /* Стиль для всего документа */
     font-size: 16pt;
     color: black;
     font-family: "Verdana"
  }
  p {                   /* Стиль для всех абзацев */
     font-size: 12pt;
     color: black;
     font-family: "Arial"
  }
  #txt {                /* Стиль для элемента с id="txt" */
     color: green
  }
  .text {               /* Стиль для класса class="text" */
     color: red
  }
```

```
 </style>
</head>
<body>
 <p>Текст 1</p><!-- Цвет по умолчанию -->
 <p id="txt">Текст 2</p><!-- Зеленый текст 12pt -->
 <p class="text">Текст 3</p><!-- Красный текст 12pt -->
 <div class="text">Текст 4</div><!-- Красный текст 16pt -->
</body>
</html>
```

Атрибуты определения стиля, указанные между тегами `<style>` и `</style>`, заключаются в фигурные скобки. Если атрибутов несколько, то они приводятся через точку с запятой:

```
<Селектор> { <Атрибут 1>: <Значение 1>; ...; <Атрибут N>: <Значение N> }
```

После последнего определения стиля перед закрывающей фигурной скобкой ставить точку с запятой не обязательно.

В листинге 2.1 мы воспользовались тремя основными видами селекторов: именем тега, идентификатором элемента (параметр `id`) и стилевым классом (параметр `class`). Селектор с именем тега применяет стили ко всем тегам с указанным именем. Разница между двумя последними способами заключается в том, что идентификатор указывается только для одного элемента, тогда как стилевой класс можно прикрепить к нескольким элементам, причем элементы могут быть разными (например, в нашем примере это и абзац, и элемент `div`).

Стили, примененные непосредственно к элементу, могут переопределять стили для всего документа. Например, стили, указанные для элемента `body`, наследуются всеми элементами. Однако для элемента с идентификатором `txt` мы явно указываем другой цвет текста, поэтому цвет, указанный в элементе `body`, переопределяется. Для стилевого класса `text` не указан размер шрифта, поэтому его значение будет наследоваться от других элементов. Для элементов `p` у нас указан размер `12pt`, который переопределит размер для элемента `body`, а вот элемент `div` наследует размер `16pt` от элемента `body`.

Чтобы наглядно увидеть процесс наследования и переопределения атрибутов стиля, воспользуемся панелью **Инструменты разработчика** Web-браузера Firefox: открываем вкладку **Инспектор**, выделяем первый абзац и переходим на вкладку **Правила**. Результат показан на рис. 2.1 — все переопределенные стили в разделе **Унаследовано от body** зачеркнуты. Если мы выделим элемент `div`, то зачеркнут будет только один атрибут `color` (рис. 2.2). Это означает, что элемент `div` наследует все стили от элемента `body`, кроме `color`.

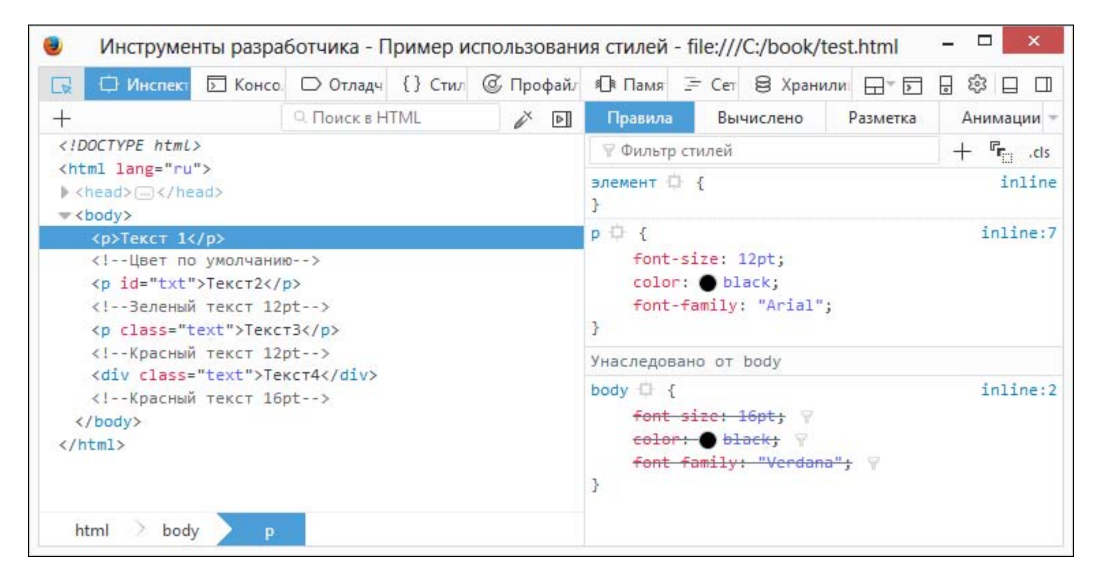


Рис. 2.1. Переопределение стилей

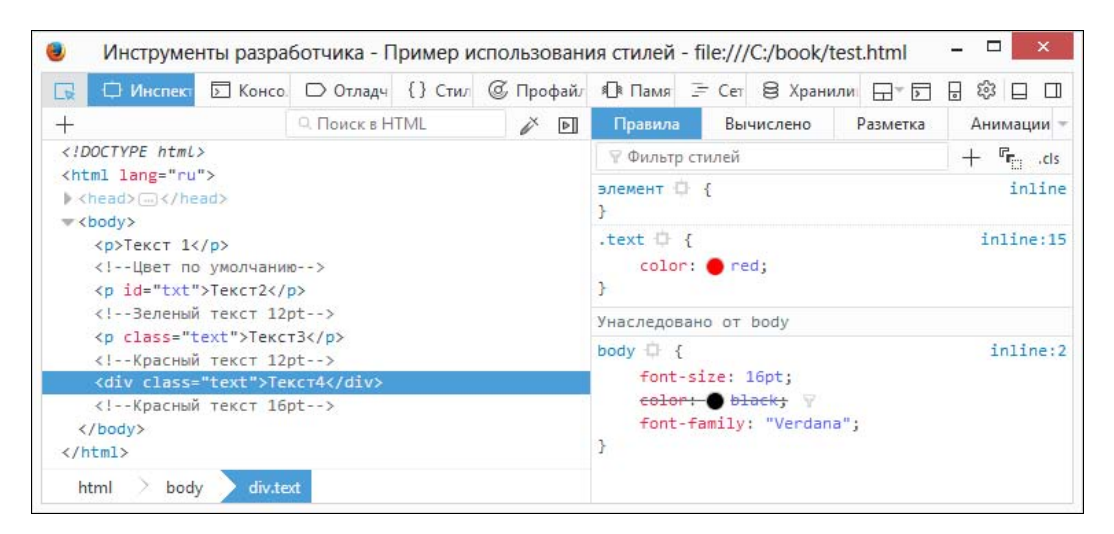


Рис. 2.2. Наследование стилей

2.1.3. Вынесение таблицы стилей в отдельный файл

Таблицу стилей можно вынести в отдельный файл. Файл с таблицей стилей обычно имеет расширение css и может редактироваться любым текстовым редактором, например Notepad++. Задать расширение файлу можно точно так же, как и при создании файла с расширением html. И кодировка файлов точно такая же, как и у HTML-документа, — в нашем случае: UTF-8 (без BOM).

Вынесем таблицу стилей в отдельный файл style.css (листинг 2.2) и подключим его к основному документу test.html (листинг 2.3). С помощью правила `@charset` зада-

дим кодировку файла (не забудьте сам файл сохранить в этой кодировке). Обратите внимание: это правило должно быть самой первой строкой в файле!

Листинг 2.2. Содержимое файла style.css

```
@charset "utf-8";     /* Кодировка файла */
/* Так можно вставить комментарий */
body {                /* Стиль для всего документа */
   font-size: 16pt;
   color: black;
   font-family: "Verdana"
}
p {                   /* Стиль для всех абзацев */
   font-size: 12pt;
   color: black;
   font-family: "Arial"
}
#txt {               /* Стиль для элемента с id="txt" */
   color: green
}
.text {              /* Стиль для класса class="text" */
   color: red
}
```

Листинг 2.3. Содержимое файла test.html

```
<!DOCTYPE html>
<html lang="ru">
<head>
 <meta charset="utf-8">
 <title>Пример использования стилей</title>
 <link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
 <p>Текст 1</p><!-- Цвет по умолчанию -->
 <p id="txt">Текст 2</p><!-- Зеленый текст 12pt -->
 <p class="text">Текст 3</p><!-- Красный текст 12pt -->
 <div class="text">Текст 4</div><!-- Красный текст 16pt -->
</body>
</html>
```

Сохраним оба файла в одной папке и откроем файл test.html в Web-браузере. Результат будет таким же, как и в предыдущем примере.

Отдельный файл с таблицей стилей прикрепляется к HTML-документу с помощью одинарного тега `<link>`. В параметре `href` указывается абсолютный или относительный интернет-адрес (URL) файла, а в параметре `rel` должно быть значение

stylesheet, показывающее, что присоединяемый таким образом документ содержит таблицу стилей:

```
<link rel="stylesheet" type="text/css" href="style.css">
```

Подключить внешний CSS-файл можно также с помощью правила @import:

```
@import url(<URL-адрес>)[ <Тип устройства>];
@import <URL-адрес>[ <Тип устройства>];
```

Правило @import должно быть расположено внутри тега <style>:

```
<style type="text/css">
  @import url("style.css");
</style>
```

В необязательном параметре <Тип устройства> можно указать устройство, для которого предназначена подключаемая таблица стилей. Например, all — для любых устройств, print — для предварительного просмотра и распечатки документа, screen — для экрана монитора:

```
<style type="text/css">
  @import "style.css" print;
</style>
```

Таблицу стилей, вынесенную в отдельный файл, можно использовать в нескольких HTML-документах.

2.1.4. Приоритет применения стилей

Предположим, что для абзаца определен атрибут color в параметре style одного цвета, в теге <style> другого цвета, а в отдельном файле (листинг 2.4) — третьего цвета. Кроме того, в теге <span> задан четвертый цвет (листинг 2.5).

Листинг 2.4. Содержимое файла style.css

```
p { color: red }
```

Листинг 2.5. Содержимое файла test.html

```
<!DOCTYPE html>
<html lang="ru">
<head>
 <meta charset="utf-8">
 <title>Приоритет применения стилей</title>
 <link rel="stylesheet" type="text/css" href="style.css">
 <style type="text/css">
   p { color: blue }
 </style>
</head>
```

```
<body>
 <p style="color: green">
    <span style="color: yellow">Текст 1</span>
 </p>
 <p style="color: green">Текст 2</p>
</body>
</html>
```

Какого цвета будет текст `Текст 1`? И какого цвета будет абзац с текстом `Текст 2`? Для ответа на эти вопросы и существует приоритет стилей:

- стиль, заданный таблицей стилей, будет отменен, если в HTML-коде явно описано форматирование элемента;
- стиль, заданный в теге `<style>`, будет отменен, если в параметре `style` тега указан другой стиль;
- стиль, заданный в отдельном файле, будет отменен, если в теге `<style>` указано другое определение стиля.

Именно из-за такой структуры приоритетов таблицы стилей и называют *каскадными*.

Иными словами, наименьший приоритет имеет стиль, описанный в отдельном файле, а самый высокий — стиль, указанный последним. В нашем примере к тексту `Текст 1` будет применено форматирование, указанное в теге `<span>`, т. е. текст будет желтого цвета. А абзац с текстом `Текст 2` будет иметь цвет, указанный в параметре `style`, т. е. зеленый.

Кроме того, следует учитывать, что стиль, заданный через идентификатор, будет иметь более высокий приоритет, чем стиль, заданный через класс:

```
<style type="text/css">
   #id1 { color: red }
   .cls1 { color: blue }
</style>
...
<p id="id1" class="cls1">Текст</p>
```

В этом примере текст абзаца будет красного цвета, а не синего.

С помощью свойства `!important` можно изменить приоритет. Для примера изменим содержимое файла style.css (листинг 2.4) на:

```
p { color: red !important }
```

В результате мы переопределили все стили, и абзац со словом `Текст 2` будет иметь красный цвет, а не зеленый. Однако текст `Текст 1` так и останется желтого цвета, т. к. цвет этот указан в теге `<span>`.

2.2. Указание значений атрибутов

После значения в определении стиля необходимо уточнить, в каких единицах указано это значение. Не указывать единицы измерения можно лишь для значения `0`.

2.2.1. Числа

Дробная часть вещественного числа указывается через символ «точка»:

```
p { color: rgba(255, 0, 0, 0.5) }
```

Если перед символом «точка» указан нуль, то его можно опустить:

```
p { color: rgba(255, 0, 0, .5) }
```

2.2.2. Размеры

Размеры в CSS можно задавать в абсолютных или относительных единицах.

Абсолютные единицы:

- `px` — пиксел;
- `mm` — миллиметр;
- `cm` — сантиметр;
- `in` — дюйм: `1 in = 2.54 cm`;
- `pt` — пункт: `1 pt = 1/72 in`;
- `pc` — пика: `1 pc = 12 pt`.

Относительные единицы:

- `%` — процент;
- `em` — высота текущего шрифта;
- `ex` — высота буквы «x» текущего шрифта.

В CSS 3 доступны также следующие относительные единицы:

- `ch` — ширина символа «0» текущего шрифта;
- `rem` — высота шрифта, указанного для корневого элемента (тега `<html>`);
- `vw` — 1% от ширины области просмотра окна Web-браузера;
- `vh` — 1% от высоты области просмотра окна Web-браузера;
- `vmin` — 1% от меньшего значения из ширины или высоты области просмотра;
- `vmax` — 1% от большего значения из ширины или высоты области просмотра.

В CSS 3 в качестве значения атрибута стиля мы можем указать результат какого-либо вычисления. Для этого предусмотрена функция `calc()`. Само выражение, которое требуется вычислить, записывается сразу же после ее имени, берется в скобки и имеет вид обычной алгебраической формулы:

```
#main { width: calc(100vw - 400px) }
```

Этот стиль устанавливает для элемента с идентификатором `main` ширину, равную ширине области просмотра окна Web-браузера, за вычетом 400 пикселов.

В выражениях, указанных в функции `calc()`, допустимы следующие операции: сложение (обозначается символом `+`), вычитание (`-`), умножение (`*`) и деление (`/`).

2.2.3. Цвет

Цвет можно задать одним из следующих способов:

- именем цвета — `blue`, `green` и т. д.:

  ```
  p { color: red }
  ```

- значением вида `#RGB`, где `R` — насыщенность красного, `G` — насыщенность зеленого и `B` — насыщенность синего в цвете. Значения задаются одинарными шестнадцатеричными числами от `0` до `F`:

  ```
  p { color: #F00 }
  ```

- значением вида `#RRGGBB`, где `RR` — насыщенность красного, `GG` — насыщенность зеленого и `BB` — насыщенность синего в цвете. В таком формате значения задаются двузначными шестнадцатеричными числами от `00` до `FF`:

  ```
  p { color: #FF0000 }
  ```

- значением вида `rgb(R, G, B)`, где `R`, `G` и `B` — насыщенности красного, зеленого и синего цветов, которые задаются десятичными числами от `0` до `255`:

  ```
  p { color: rgb(255, 0, 0) }
  ```

- значением вида `rgb(R%, G%, B%)`, где `R%`, `G%` и `B%` — насыщенности красного, зеленого и синего цветов, которые задаются в процентах:

  ```
  p { color: rgb(100%, 0%, 0%) }
  ```

Все приведенные здесь примеры задают красный цвет.

В CSS 3 добавлены следующие способы задания цвета:

- значением вида `rgba(R, G, B, A)`, где `R`, `G` и `B` — насыщенности красного, зеленого и синего цветов, которые задаются десятичными числами от `0` до `255`, `A` — задает уровень прозрачности цвета (альфа-канал), представляющий собой значение от `0.0` (цвет полностью прозрачен) до `1.0` (цвет полностью непрозрачен). Вот пример указания полупрозрачного красного цвета:

  ```
  p { color: rgba(255, 0, 0, 0.5) }
  ```

  Значения насыщенности красного, зеленого и синего цветов можно также задать в процентах:

  ```
  p { color: rgba(100%, 0%, 0%, 0.5) }
  ```

- значением вида `hsl(H, S, L)`, где `H` — оттенок (число от `0` до `359`), `S` — насыщенность (проценты от `0` до `100`) и `L` — светлота (проценты от `0` до `100`):

  ```
  p { color: hsl(0, 100%, 50%) }
  ```

- значением вида hsla(H, S, L, A), где H — оттенок (число от 0 до 359), S — насыщенность (проценты от 0 до 100), L — светлота (проценты от 0 до 100) и A — уровень прозрачности цвета (альфа-канал), представляющий собой значение от 0.0 (цвет полностью прозрачен) до 1.0 (цвет полностью непрозрачен):

```
p { color: hsla(0, 100%, 50%, 0.5) }
```

Приведем названия наиболее часто используемых цветов:

- black — #000000 — черный;
- white — #FFFFFF — белый;
- yellow — #FFFF00 — желтый;
- silver — #C0C0C0 — серый;
- red — #FF0000 — красный;
- green — #008000 — зеленый;
- gray — #808080 — темно-серый;
- blue — #0000FF — синий;
- navy — #000080 — темно-синий;
- purple — #800080 — фиолетовый.

2.2.4. Строки

Строки в CSS указываются внутри двойных или одинарных кавычек. С помощью символа «\» можно экранировать кавычку внутри строки:

```
p { font-family: "Times New Roman" }
div { font-family: 'Arial' }
p:before { content: 'Д\'Арк ' }
```

2.2.5. Углы

Для обозначения углов в CSS 3 используются следующие единицы:

- deg — градусы:

```
#mydiv { transform: rotate(45deg) }
```

- rad — радианы:

```
#mydiv { transform: rotate(0.79rad) }
```

- grad — грады:

```
#mydiv { transform: rotate(50grad) }
```

- turn — повороты (полный круг равен 1turn):

```
#mydiv { transform: rotate(0.125turn) }
```

Пример вращения элемента с идентификатором mydiv на угол 45 градусов:

```
#mydiv {
   position: absolute;
   left: 200px; top: 100px;
   width: 100px; height: 100px;
   background: red;
   transform: rotate(45deg)
}
...
<div id="mydiv"></div>
```

Перед значением угла можно указать знак минус. В этом случае вращение будет выполняться против часовой стрелки:

```
#mydiv { transform: rotate(-45deg) }
```

2.2.6. Универсальные значения

В качестве значения атрибутов можно указать следующие ключевые слова:

- inherit — значение наследуется от родителя:

  ```
  background-color: inherit
  ```

- initial — задает исходное значение, используемое Web-браузером по умолчанию:

  ```
  background-color: initial
  ```

2.3. CSS-селекторы

Атрибуты определения стиля, указанные между тегами <style> и </style> или в отдельном файле, заключаются в фигурные скобки. Если атрибутов несколько, то они приводятся через точку с запятой в формате:

```
<Селектор> { <Атрибут 1>: <Значение 1>; ...; <Атрибут N>: <Значение N> }
```

2.3.1. Основные селекторы

В параметре <Селектор> могут быть указаны следующие селекторы:

- * — все теги. Уберем все внешние и внутренние отступы:

  ```
  * { margin: 0; padding: 0 }
  ```

- Тег — все теги, имеющие указанное имя:

  ```
  p { font-size: 12pt; color: green; font-family: "Arial" }
  ...
  <p>Текст2</p><!-- Зеленый текст -->
  ```

- `.Класс` — все теги, имеющие указанный класс:

  ```
  .text1 { font-size: 12pt; color: red; font-family: "Arial" }
  ...
  <div class="text1">Текст1</div><!-- Красный текст -->
  <p class="text1">Текст2</p><!-- Красный текст -->
  ```

  И `Текст1` и `Текст2` будут красного цвета, хотя они находятся в разных тегах;

- `Тег.Класс` — все теги, имеющие указанное имя и класс:

  ```
  p.text2 { font-size: 12pt; color: blue }
  ...
  <p class="text2">Текст1</p><!-- Синий текст -->
  ```

  Обратите внимание, что если имя класса указать в другом теге, то стиль не будет применен к этому тегу:

  ```
  <div class="text2">Текст2</div>
  ```

  В этом случае фрагмент текста `Текст2` не будет отображен синим цветом, т. к. имя класса `text2` применяется только к тегу `<p>`;

- `#Идентификатор` — элемент с указанным идентификатором:

  ```
  #txt1 { color: red }
  ...
  <p id="txt1">Текст</p>
  ```

Стили можно привязать сразу к нескольким селекторам, в этом случае селекторы указываются через запятую:

```
p, div { font-family: "Arial" }
```

Привязаться к другим элементам можно следующими способами:

- `Селектор1 Селектор2` — все элементы, соответствующие параметру `Селектор2`, которые располагаются внутри контейнера, соответствующего параметру `Селектор1`:

  ```
  div a { color: red }
  ```

  Цвет текста ссылки станет красным, если тег `<a>` находится внутри тега `<div>`:

  ```
  <div><a href="link.html">Ссылка</a></div>
  ```

- `Селектор1 > Селектор2` — все элементы, соответствующие параметру `Селектор2`, которые являются дочерними для контейнера, соответствующего параметру `Селектор1`:

  ```
  div > a { color: red }
  ```

  Цвет текста ссылки станет красным, если тег `<a>` находится внутри тега `<div>` и не вложен в другой тег:

  ```
  <div>
  <a href="link1.html">Ссылка 1</a><br>
  <span><a href="link2.html">Ссылка 2</a></span>
  </div>
  ```

В этом примере только первая ссылка станет красного цвета, т. к. вторая ссылка расположена внутри тега `<span>`;

- `Селектор1 + Селектор2` — элемент, соответствующий параметру `Селектор2`, который является соседним для элемента, соответствующего параметру `Селектор1`, и следует сразу после него:

  ```
  div + p { color: red }
  ```

  Цвет текста абзаца станет красным, если тег `<p>` следует сразу после элемента `div`:

  ```
  <div>Текст</div><p>Текст</p>
  ```

- `Селектор1 ~ Селектор2` — элемент, соответствующий параметру `Селектор2`, который является соседним для элемента, соответствующего параметру `Селектор1`, и следует после него, причем необязательно непосредственно:

  ```
  div ~ h6 { color: red }
  ```

  Цвет текста заголовка станет красным, если тег `<h6>` следует за элементом `div` и, возможно, отделяется от него другими элементами:

  ```
  <div>Текст</div>
  <p>Тоже текст</p>
  <h6>Красный заголовок</h6>
  ```

При необходимости можно составлять выражения из нескольких селекторов:

```
div span a { color: red }
```

Цвет текста ссылки станет красным, если тег `<a>` расположен внутри тега `<span>`, а тот в свою очередь вложен в тег `<div>`:

```
<div>
  <a href="link1.html">Ссылка 1</a><br>
  <span>
    <a href="link2.html">Ссылка 2</a><br>
  </span>
</div>
```

В этом примере только `Ссылка 2` будет красного цвета.

2.3.2. Привязка к параметрам тегов

Для привязки к параметрам тегов применяются следующие селекторы:

- `[Параметр]` — элементы с указанным параметром:

  ```
  a[id] { color: red }
  ```

  Цвет текста ссылки станет красным, если тег `<a>` имеет параметр `id`:

  ```
  <a id="link1" href="link1.html">Ссылка 1</a>
  ```

- `[Параметр='Значение']` — элементы, у которых параметр точно равен значению:

  ```
  a[href="link1.html"] { color: red }
  ```

Цвет текста ссылки станет красным, если параметр `href` тега `<a>` имеет значение `"link1.html"`;

- `[Параметр^='Значение']` — элементы, у которых значение параметра начинается с указанного значения:

```
a[href^="li"] { color: red }
```

Цвет текста ссылки станет красным, если значение параметра `href` тега `<a>` начинается с `li`;

- `[Параметр$='Значение']` — элементы, у которых значение параметра оканчивается указанным значением:

```
a[href$=".html"] { color: red }
```

Цвет текста ссылки станет красным, если значение параметра `href` тега `<a>` оканчивается на `.html`;

- `[Параметр*='Значение']` — элементы, у которых параметр содержит указанное значение:

```
a[href*="link"] { color: red }
```

Цвет текста ссылки станет красным, если значение параметра `href` тега `<a>` содержит фрагмент `link`;

- `[Параметр~='Слово']` — элементы, у которых параметр содержит указанное слово целиком (используется когда значением параметра являются слова, разделенные пробелами):

```
a[class~="class2"] { color: red }
```

Цвет текста ссылки станет красным, если параметр `class` тега `<a>` содержит слово `class2`:

```
<a class="class1 class2 class3" href="link1.html">Ссылка</a>
```

- `[Параметр|='Значение']` — элементы, у которых параметр точно равен значению или начинается с указанного значения после которого следует дефис:

```
a[class|="class1"] { color: red }
```

Цвет текста ссылки станет красным, если значение параметра `class` тега `<a>` равно `class1` или после `class1` идет дефис и любое другое значение:

```
<a class="class1" href="link1.html">Ссылка 1</a>
<a class="class1-new" href="link2.html">Ссылка 2</a>
```

В этом примере обе ссылки будут красного цвета.

2.3.3. Псевдоэлементы

В качестве селектора могут быть также указаны следующие псевдоэлементы:

- `::first-letter` — задает стиль для первой буквы. Выделим первую букву всех абзацев:

```
p::first-letter { font-size: 150%; font-weight: bold;
                  color: red }
```

- ::first-line — задает стиль для первой строки:

```
p::first-line { font-weight: bold; color: red }
...
<p>Красный полужирный текст<br>
Обычный текст</p>
```

- ::before и ::after — позволяют добавить текст в начало и в конец элемента соответственно. Добавляемый текст должен быть указан в атрибуте content:

```
p::before { content: "before " }
p::after { content: " after" }
...
<p>Текст</p>
```

Результат:

```
<p>before Текст after</p>
```

Атрибут content может принимать следующие значения:

- normal и none — не добавляют содержимого;
- строка — задает выводимый текст:

```
p::before { content: "before " }
```

- attr(<Параметр>) — возвращает значение указанного параметра или пустую строку. Выведем после текста ссылки URL-адрес внутри круглых скобок:

```
a::after { content: "(" attr(href) ")"; margin-left: 1em }
```

- url(<URL-адрес>) — задает URL-адрес вставляемого объекта (например, изображения):

```
p::before { content: url(marker.png); margin-right: 5px }
```

- open-quote — вставляет открывающую кавычку;
- close-quote — вставляет закрывающую кавычку. Тип кавычек задается с помощью атрибута quotes:

```
p { quotes: '"' '"' }
p::before { content: open-quote; color: red }
p::after { content: close-quote; color: red }
```

- no-open-quote — отменяет вставку открывающей кавычки;
- no-close-quote — отменяет вставку закрывающей кавычки:

```
q::before { content: no-open-quote }
q::after { content: no-close-quote }
```

- counter(<Переменная>) — вставляет значение счетчика. Добавим перед каждым абзацем фрагмент "Абзац ", номер абзаца и фрагмент ":":

```
body { counter-reset: mycounter }
p::before { counter-increment: mycounter;
            content: "Абзац " counter(mycounter) ":";
            margin-right: 5px }
```

Для управления счетчиком предназначены следующие атрибуты:

- `counter-reset` — создает переменную и присваивает ей начальное значение:

  ```
  counter-reset: <Переменная>[ <Начальное значение>]
  counter-reset: none
  ```

  Если начальное значение не задано, то используется значение `0`:

  ```
  body { counter-reset: mycounter }
  ```

- `counter-increment` — увеличивает значение переменной на единицу или на указанную величину:

  ```
  counter-increment: <Переменная>[ <Величина приращения>]
  counter-increment: none
  ```

  Пример:

  ```
  p::before { counter-increment: mycounter;
              content: "Абзац " counter(mycounter) ":";
              margin-right: 5px }
  ```

Существуют также следующие нестандартные псевдоэлементы:

- `::placeholder` — позволяет применить стили к тексту подсказки внутри поля:

  ```
  input[type="text"]::-webkit-input-placeholder { color: red }
  input[type="text"]:-ms-input-placeholder { color: red }
  input[type="text"]::-ms-input-placeholder { color: red }
  input[type="text"]::-moz-placeholder { color: red }
  input[type="text"]::placeholder { color: red }
  ...
  <input type="text" placeholder="Текст подсказки">
  ```

  В этом примере мы воспользовались «вендорными» атрибутами, т. к. не все Web-браузеры поддерживают псевдоэлемент `::placeholder`. Обратите внимание: вначале указываются «вендорные» атрибуты, а лишь затем стандартный атрибут. Кроме того, нельзя объединять разные «вендорные» атрибуты в одном определении стиля. Поэтому мы несколько раз продублировали одинаковое определение стиля с разными селекторами. «Вендорные» атрибуты Web-браузера Internet Explorer имеют префикс `-ms-`, Chrome и Safari — префикс `-webkit-`, Firefox — префикс `-moz-`;

- `::selection` — задает стиль для выделенного фрагмента:

  ```
  ::-moz-selection { background: green }
  ::selection { background: green }
  ...
  <p>Текст</p>
  ```

  В этом примере выделенный текст будет иметь зеленый цвет фона.

ПРИМЕЧАНИЕ

В CSS 2 названия псевдоэлементов начинались с двоеточия. В CSS 3 они начинаются с двойного двоеточия, чтобы их можно было отличать от названий псевдоклассов.

2.3.4. Псевдоклассы

В CSS 3 качестве селектора могут быть также указаны следующие псевдоклассы:

- `:empty` — пустой элемент, не имеющий никакого содержимого. Сделаем все пустые абзацы невидимыми:

```
p:empty { display: none }
```

- `:first-child` — привяжет стиль к элементу, только если он относится к указанному типу и является первым дочерним элементом в контейнере. Увеличим размер шрифта у абзацев, которые являются первыми элементами в контейнере:

```
p:first-child { font-size: larger }
...
<div>
  <p>Этот абзац будет выведен увеличенным шрифтом, поскольку он
  является первым элементом в контейнере.</p>
  <p>А этот будет выведен шрифтом стандартного размера, поскольку
  он не является первым элементом в контейнере.</p>
</div>
<div>
  <h1>Заголовок</h1>
  <p>Этот абзац будет выведен шрифтом стандартного размера,
  поскольку он не является первым элементом в контейнере.</p>
</div>
```

- `:first-of-type` — задает стиль для первого элемента в контейнере, относящегося к заданному типу. Увеличиваем размер шрифта у всех абзацев — первых потомков контейнеров:

```
p:first-of-type { font-size: larger }
...
<div>
  <p>Этот абзац будет выведен увеличенным шрифтом, поскольку он
  является первым абзацем в контейнере.</p>
  <p>А этот будет выведен шрифтом стандартного размера.</p>
</div>
<div>
  <h1>Заголовок</h1>
  <p>Этот абзац будет выведен увеличенным шрифтом, поскольку он
  является первым абзацем в контейнере.</p>
</div>
```

- `:last-child` — привяжет стиль к элементу, только если он относится к указанному типу и является последним дочерним элементом в контейнере;

- `:last-of-type` — задает стиль для последнего элемента в контейнере, относящегося к заданному типу;
- `:nth-child(<N>)` — привяжет стиль к элементу, только если он относится к указанному типу и является точно `N`-м по счету элементом в контейнере (нумерация элементов начинается с `1`). Увеличим размер шрифта у абзацев, которые являются вторыми потомками контейнеров:

```
p:nth-child(2) { font-size: larger }
...
<div>
  <p>Этот абзац будет выведен шрифтом стандартного размера.</p>
  <p>А этот будет выведен увеличенным шрифтом.</p>
</div>
<div>
  <h1>Заголовок</h1>
  <p>Этот абзац также будет выведен увеличенным шрифтом.</p>
</div>
```

  Вместо числа можно указать значения `odd` (все нечетные номера), `even` (все четные номера) или выражение `An+B`. Например, выражение `2n` является аналогом значения `even`, а выражение `2n+1` — аналогом значения `odd`;
- `:nth-last-child(<N>)` — привяжет стиль к элементу, только если он относится к указанному типу и является точно `N`-м по счету элементом в контейнере, считая с конца;
- `:nth-of-type(<N>)` — задает стиль для `N`-го элемента в контейнере, относящегося к указанному типу. Увеличим размер шрифта у каждого второго абзаца в контейнере:

```
p:nth-of-type(2) { font-size: larger }
...
<div>
  <p>Этот абзац будет выведен шрифтом стандартного размера.</p>
  <p>А этот будет выведен увеличенным шрифтом.</p>
</div>
<div>
  <h1>Заголовок</h1>
  <p>Этот абзац будет выведен шрифтом стандартного размера.</p>
  <p>А этот будет выведен увеличенным шрифтом.</p>
</div>
```

- `:nth-last-of-type(<N>)` — задает стиль для `N`-го элемента указанного типа в контейнере, считая с конца;
- `:only-child` — привяжет стиль к элементу, только если он относится к указанному типу и является единственным элементом в контейнере. Увеличим размер шрифта у абзаца, который является единственным элементом в контейнере:

```
p:only-child { font-size: larger }
```

```
...
<div>
  <p>Этот абзац будет выведен увеличенным шрифтом.</p>
</div>
<div>
  <p>А этот будет выведен шрифтом стандартного размера.</p>
  <p>И этот тоже.</p>
</div>
<div>
  <h1>Заголовок</h1>
  <p>И этот абзац будет выведен шрифтом стандартного размера.</p>
</div>
```

- `:only-of-type` — задает стиль для единственного элемента указанного типа в контейнере. Увеличим размер шрифта у каждого единственного абзаца в контейнере (при этом там могут находиться элементы других типов):

```
p:only-of-type { font-size: larger }
...
<div>
  <p>Этот абзац будет выведен увеличенным шрифтом.</p>
</div>
<div>
  <p>А этот будет выведен шрифтом стандартного размера.</p>
  <p>И этот тоже.</p>
</div>
<div>
  <h1>Заголовок</h1>
  <p>Этот абзац будет выведен увеличенным шрифтом.</p>
</div>
```

- `:target` — «якорь», к которому был выполнен переход по внутренней гиперссылке. Сделаем цвет текста целевого элемента красным:

```
:target { color: red }
...
<div>
  <h1 id="header">Заголовок</h1>
  <p id="txt">Текст абзаца</p>
  <a href="#header">Перейти к элементу header</a>
  <a href="#txt">Перейти к элементу txt</a>
</div>
```

  Попробуйте перейти по гиперссылкам, и цвет текста заголовка или абзаца станет красным;

- `:root` — корневой элемент документа;
- `:link` — непосещенная гиперссылка:

```
a:link { color: #000000 }
```

- `:visited` — посещенная гиперссылка:

```
a:visited { color: #000080 }
```

- `:active` — активная гиперссылка или другой элемент, над которым была нажата и удерживается нажатой кнопка мыши:

```
a:active { color: #FF0000 }
p:active { color: #FF0000 }
```

- `:hover` — гиперссылка, на которую указывает курсор мыши или другой элемент, над которым находится указатель мыши:

```
a:hover { color: green; text-decoration: none }
p:hover { color: #FF0000 }
```

- `:focus` — гиперссылка или элемент управления, имеющий фокус ввода. При получении фокуса ввода сделаем фон текстового поля зеленым, а цвет текста белым:

```
input[type="text"]:focus { background: green; color: white }
...
<input type="text">
```

- `:enabled` — элемент управления, доступный для пользователя;
- `:disabled` — элемент управления, недоступный для пользователя (тег которого содержит параметр `disabled`). Зададим для текста недоступных элементов управления серый цвет:

```
input:disabled, select:disabled, textarea:disabled {
   color: #cccccc }
```

- `:read-only` — элемент управления, доступный только для чтения (тег которого содержит параметр `readonly`):

```
input:-moz-read-only { background-color: #cccccc }
input:read-only { background-color: #cccccc }
```

  Этот псевдокласс не поддерживается браузером Internet Explorer.

- `:read-write` — элемент управления, доступный для чтения и записи (тег которого не содержит параметра `readonly`):

```
input:-moz-read-write { background-color: #cccccc }
input:read-write { background-color: #cccccc }
```

  Этот псевдокласс не поддерживается браузером Internet Explorer.

- `:required` — обязательный элемент управления (тег которого имеет параметр `required`);
- `:optional` — необязательный элемент управления (тег которого не имеет параметра `required`);
- `:valid` — элемент управления, в котором указано корректное значение;

- `:invalid` — элемент управления, в котором указано некорректное значение:

  ```
  input:invalid { color: red }
  ```

- `:in-range` — поле ввода числового значения или регулятор, в котором указано корректное значение, укладывающееся в заданный диапазон;
- `:out-of-range` — поле ввода числового значения или регулятор, в котором указано некорректное значение, не укладывающееся в заданный диапазон:

  ```
  input:out-of-range { color: red }
  ...
  <input type="number" min="0" max="10">
  ```

- `:checked` — установленный флажок или переключатель:

  ```
  input[type="checkbox"]:checked + span { color: green }
  ...
  <input type="checkbox" checked><span>Текст</span>
  ```

- `:default` — элемент управления по умолчанию (например, кнопка отправки формы);
- `:indeterminate` — элемент управления в неопределенном состоянии:

  ```
  input:indeterminate + span { color: gray }
  ...
  <input type="checkbox" id="check1"><span>Текст</span>
  <script>
  document.getElementById('check1').indeterminate = true;
  </script>
  ```

Псевдокласс `:not` стоит особняком. Он позволяет привязать стиль к любому элементу страницы, не удовлетворяющему заданным условиям. Вот формат записи этого псевдокласса:

```
<Основной селектор>:not(<Селектор выбора>)
```

Стиль будет привязан к элементу Web-страницы, удовлетворяющему основному селектору и не удовлетворяющему селектору выбора:

```
p:not(:first-child) { font-size: larger }
```

Этот стиль будет применен лишь к абзацам, не являющимся первыми дочерними элементами в контейнерах.

2.4. Форматирование шрифта

Каскадные таблицы стилей позволяют задать название и размер шрифта, его стиль и «жирность», а также цвет текста. Кроме того, можно указать несколько имен шрифтов и одно из названий альтернативных семейств — ведь на компьютере пользователя может не быть нужного шрифта.

2.4.1. Имя шрифта

Имя шрифта позволяет задать атрибут `font-family`:

```
p { font-family: "Arial" }
```

В ряде случаев шрифт может отсутствовать на компьютере пользователя. Поэтому лучше указывать несколько альтернативных шрифтов. Имена шрифтов при этом указываются через запятую:

```
p { font-family: "Verdana", "Tahoma" }
```

Можно также указать одно из пяти типовых семейств шрифтов: `serif`, `sans-serif`, `cursive`, `fantasy` или `monospace`:

```
p { font-family: "Verdana", "Tahoma", sans-serif }
```

2.4.2. Стиль шрифта

Стиль шрифта позволяет задать атрибут `font-style`. Он может принимать следующие значения:

- `normal` — нормальный шрифт:

  ```
  p { font-family: "Arial"; font-style: normal }
  ```

- `italic` — курсивный шрифт:

  ```
  p { font-family: "Arial"; font-style: italic }
  ```

- `oblique` — наклонный шрифт:

  ```
  p { font-family: "Arial"; font-style: oblique }
  ```

2.4.3. Размер шрифта

Размер шрифта позволяет задать атрибут `font-size`:

```
.text1 { font-size: 12pt; font-family: "Arial" }
```

Можно указать абсолютную величину или одну из типовых констант: `xx-small`, `x-small`, `small`, `medium`, `large`, `x-large` или `xx-large`:

```
.text1 { font-size: large; font-family: "Arial" }
```

Кроме того, можно указать относительную величину (например, значение в процентах) или одну из двух констант: `larger` или `smaller`:

```
.text1 { font-size: 150%; font-family: "Arial" }
.text2 { font-size: smaller; font-family: "Arial" }
```

2.4.4. Цвет текста

Цвет текста позволяет задать атрибут `color`:

```
.text1 { font-size: 12pt; font-family: "Arial"; color: red }
.text2 { font-size: 12pt; font-family: "Arial"; color: #00FF00 }
.text3 { font-size: 12pt; font-family: "Arial"; color: rgb(255, 0, 0) }
```

Можно также указать значение `transparent`, означающее прозрачный цвет:

```
.text1 { color: transparent }
```

2.4.5. Жирность шрифта

Управлять жирностью шрифта позволяет атрибут `font-weight`. Он может принимать следующие значения:

- `100`, `200`, `300`, `400`, `500`, `600`, `700`, `800`, `900` — значение `100` соответствует самому бледному шрифту, а `900` — самому жирному:

  ```
  p { font-family: "Arial"; font-style: italic; font-weight: 700 }
  ```

- `normal` — нормальный шрифт. Соответствует значению `400`:

  ```
  p { font-family: "Arial"; font-weight: normal }
  ```

- `bold` — полужирный шрифт. Соответствует значению `700`:

  ```
  p { font-family: "Arial"; font-weight: bold }
  ```

- `lighter` — менее жирный, чем у родительского элемента:

  ```
  p { font-family: "Arial"; font-weight: lighter }
  ```

- `bolder` — более жирный, чем у родительского элемента:

  ```
  p { font-family: "Arial"; font-weight: bolder }
  ```

2.4.6. Вид строчных букв

Вид строчных букв задает атрибут `font-variant`. Он может принимать следующие значения:

- `normal` — значение по умолчанию:

  ```
  p { font-family: "Arial"; font-variant: normal }
  ```

- `small-caps` — строчные буквы отображаются прописными буквами уменьшенного размера:

  ```
  p { font-family: "Arial"; font-variant: small-caps }
  ```

2.4.7. Одновременное указание характеристик шрифта

Задать все характеристики шрифта одновременно позволяет атрибут `font`:

```
font: [font-style ][font-variant ][font-weight ][font-stretch ]
      font-size[/line-height] font-family
```

Параметр `line-height` задает вертикальное расстояние между базовыми линиями двух строк, а параметр `font-stretch` — ширину букв шрифта:

```
p { font: italic small-caps bold 12pt/normal "Arial" }
```

Обязательными являются только параметры `font-size` и `font-family`:

```
p { font: 12pt "Verdana", "Tahoma", sans-serif }
```

2.4.8. Загружаемые шрифты

Помимо шрифтов, установленных на компьютере, и типовых семейств шрифтов, для вывода текста на страницах мы можем использовать загружаемые шрифты. Такой шрифт хранится в виде файла на Web-сервере и загружается самим Web-браузером.

Загружаемый шрифт указывается с помощью правила `@font-face`:

```
@font-face {
  font-family: <Имя, под которым загружаемый шрифт будет доступен в
               таблице стилей>;
  src: url("<URL-адрес файла со шрифтом>")
}
```

Пример:

```
@font-face {
  font-family: MyFont;
  src: url("/fonts/myfont.ttf")
}
```

Загружаем шрифт, хранящийся в файле `myfont.ttf`, который находится в папке `fonts` в корне сайта, и задаем для него имя `MyFont`. После этого мы можем использовать загруженный шрифт где угодно, указав заданное для него имя и альтернативный шрифт или семейство шрифтов для Web-браузеров, не сумевших загрузить шрифт:

```
p { font-family: MyFont, sans-serif }
```

Все современные Web-браузеры поддерживают загружаемые шрифты, сохраненные в форматах TTF и WOFF. С другими форматами лучше не связываться, т. к. они либо поддерживаются не всеми Web-браузерами, либо при их использовании могут возникнуть проблемы.

2.5. Форматирование текста

Для текстовых фрагментов, кроме указания характеристик шрифтов, можно задать некоторые дополнительные параметры: расстояние между символами, словами и строками, вертикальное и горизонтальное выравнивание, отступ первой строки.

2.5.1. Расстояние между символами в словах

Расстояние между символами текста задает атрибут `letter-spacing`. Он может принимать следующие значения:

- `normal` — значение по умолчанию:

  ```
  p { letter-spacing: normal; font-weight: normal }
  ```

- величина в поддерживаемых CSS единицах:

  ```
  p { font-size: 12pt; color: red; letter-spacing: 5mm }
  ```

2.5.2. Расстояние между словами

Расстояние между словами задает атрибут `word-spacing`. Он может принимать следующие значения:

- `normal` — значение по умолчанию:

  ```
  p { word-spacing: normal; font-weight: normal }
  ```

- величина в поддерживаемых CSS единицах:

  ```
  p { font-size: 12pt; color: red; word-spacing: 5mm }
  ```

2.5.3. Отступ первой строки

Отступ для «красной строки» задает атрибут `text-indent`. Может задаваться абсолютная или относительная величина отступа:

```
p { text-indent: 10mm; font-style: italic; font-weight: normal }
```

2.5.4. Вертикальное расстояние между строками

Вертикальное расстояние между базовыми линиями двух строк задает атрибут `line-height`. Он может принимать следующие значения:

- `normal` — значение по умолчанию:

  ```
  p { line-height: normal; font-weight: normal }
  ```

- величина в поддерживаемых CSS единицах:

  ```
  p { font-size: 12pt; color: red; font-family: "Arial";
      line-height: 5mm }
  ```

2.5.5. Горизонтальное выравнивание текста

Горизонтальное выравнивание текста задает атрибут `text-align`. Он может принимать следующие значения:

- `center` — выравнивание по центру:

  ```
  <p style="text-align: center">Абзац с выравниванием
  по центру</p>
  ```

- `left` — выравнивание по левому краю:

  ```
  <p style="text-align: left">Абзац с выравниванием
  по левому краю</p>
  ```

- `right` — выравнивание по правому краю:

  ```
  <p style="text-align: right">Абзац с выравниванием
  по правому краю</p>
  ```

- `justify` — выравнивание по ширине (по двум сторонам):

  ```
  <p style="text-align: justify">Абзац с выравниванием
  по ширине</p>
  ```

Горизонтальное выравнивание последней строки текста задает атрибут `text-align-last`. Он может принимать следующие значения:

- `auto` — совпадает со значением атрибута `text-align`, но при значении `justify` выравнивание последней строки будет по началу блока;
- `start` — по началу блока (зависит от направления текста);
- `end` — по концу блока (зависит от направления текста);
- `left` — по левому краю;
- `right` — по правому краю;
- `center` — по центру;
- `justify` — по ширине (по двум сторонам). Если последняя строка содержит одно слово, то выполняется выравнивание по началу блока.

Пример:

```
p { width: 200px;
    text-align: justify; text-align-last: center }
...
<p>Атрибут text-align-last задает выравнивание последней строки</p>
```

2.5.6. Вертикальное выравнивание текста

Вертикальное выравнивание текста относительно элемента-родителя — например, ячейки таблицы — задает атрибут `vertical-align`. Он может принимать следующие значения:

- `baseline` — по базовой линии:

  ```
  td { font-size: 12pt; color: red; vertical-align: baseline }
  ```

- `middle` — по центру:

  ```
  td { font-size: 12pt; color: red; vertical-align: middle }
  ```

- `top` — по верху:

  ```
  td { font-size: 12pt; color: red; vertical-align: top }
  ```

- `bottom` — по низу:

  ```
  td { font-size: 12pt; color: red; vertical-align: bottom }
  ```

Атрибут `vertical-align` работает со строчными элементами и ячейками таблицы, поэтому если попробовать выполнить вертикальное выравнивание текста внутри элемента `div` или другого блочного элемента, то ничего не получится.

Со строчными элементами можно также использовать следующие значения:

- `sub` — используется для создания нижних индексов (размер шрифта не уменьшается):

```
<p>
H<span style="vertical-align: sub; font-size: 0.7em">2</span>O
</p>
```

- `super` — используется для создания верхних индексов (размер шрифта не уменьшается):

```
<p>
m<span style="vertical-align: super; font-size: 0.7em">3</span>
</p>
```

- `text-top` — выравнивание по верху текстовой строки;
- `text-bottom` — выравнивание по низу текстовой строки:

```
<p>
<img alt="" src="foto.gif" height="100"
     style="border: 1px red solid">
Строка
<span style="vertical-align: text-top; font-size: 0.6em">
text-top</span>
<span style="vertical-align: text-bottom; font-size: 0.6em">
text-bottom</span>
<span style="vertical-align: top; font-size: 0.6em">top</span>
<span style="vertical-align: bottom; font-size: 0.6em">
bottom</span>
<span style="vertical-align: middle; font-size: 0.6em">
middle</span>
Строка</p>
```

- величина в поддерживаемых CSS единицах:

```
<p>Строка
<span style="vertical-align: 20px; font-size: 0.6em">
20px</span>
<span style="vertical-align: -120%; font-size: 0.6em">
-120%</span>
Строка</p>
```

2.5.7. Подчеркивание, надчеркивание и зачеркивание текста

Подчеркнуть, надчеркнуть или зачеркнуть текст позволяет атрибут `text-decoration`. Он может принимать следующие значения:

- `none` — обычный текст (по умолчанию):

```
<p style="text-decoration: none">Текст</p>
```

- `underline` — подчеркивает текст:

```
<p style="text-decoration: underline">Подчеркнутый текст</p>
```

- `overline` — проводит линию над текстом:

```
<p style="text-decoration: overline">Надчеркнутый текст</p>
```

- `line-through` — зачеркивает текст:

```
<p style="text-decoration: line-through">Зачеркнутый текст</p>
```

В некоторых Web-браузерах можно указать дополнительные атрибуты:

- `text-decoration-color` — задает цвет линии:

```
p { text-decoration: underline; text-decoration-color: red }
```

- `text-decoration-line` — тип линии: `none`, `underline`, `overline` или `line-through`:

```
p { text-decoration-line: underline; text-decoration-color: red }
```

- `text-decoration-style` — стиль линии:
  - `solid` — линия отображается сплошной линией;
  - `dotted` — пунктирная линия;
  - `dashed` — штриховая линия;
  - `double` — двойная линия;
  - `wavy` — волнистая линия.

  Пример:

```
p {
   text-decoration-line: underline;
   text-decoration-color: red;
   text-decoration-style: double
}
```

В атрибуте `text-decoration` можно указать сразу несколько значений:

```
text-decoration: text-decoration-line text-decoration-style
                 text-decoration-color
```

Пример:

```
p { text-decoration: underline red wavy }
```

2.5.8. Изменение регистра символов

Изменить регистр символов позволяет атрибут `text-transform`. Он может принимать следующие значения:

- `capitalize` — делает первую букву каждого слова прописной;
- `uppercase` — преобразует все буквы в прописные;

- `lowercase` — преобразует все буквы в строчные;
- `none` — без преобразования.

Пример:

```
<h1 style="text-transform: capitalize">заголовок из
нескольких слов</h1>
<h1 style="text-transform: uppercase">заголовок2</h1>
<h1 style="text-transform: lowercase">ЗАГОЛОВОК3</h1>
```

Результат:

```
Заголовок Из Нескольких Слов
ЗАГОЛОВОК2
заголовок3
```

2.5.9. Обработка пробелов между словами

Установить тип обработки пробелов позволяет атрибут `white-space`. По умолчанию несколько пробелов подряд выводятся в окне Web-браузера как один пробел. Атрибут может принимать следующие значения:

- `normal` — текст выводится стандартным образом:

  ```
  <p style="white-space: normal">
  Строка          1
  Строка 2
  </p>
  ```

  Результат в окне Web-браузера:

  ```
  Строка 1 Строка 2
  ```

- `pre` — сохраняются все пробелы и переносы строк. Текст автоматически на новую строку не переносится:

  ```
  <p style="white-space: pre">
  Строка          1
  Строка 2
  </p>
  ```

  Результат в окне Web-браузера:

  ```
  Строка          1
  Строка 2
  ```

- `nowrap` — переносы строк в HTML-коде игнорируются. Текст автоматически на новую строку не переносится. Но если внутри строки содержится тег `<br>`, то он вставляет перенос строки:

  ```
  <p style="white-space: nowrap">
  Строка          1
  Строка 2<br>
  Строка 3
  </p>
  ```

Результат в окне Web-браузера:

```
Строка 1 Строка 2
Строка 3
```

- `pre-line` — переносы строк сохраняются, а пробелы игнорируются. Если текст не помещается на строке, то он будет автоматически перенесен на новую строку:

```
<p style="white-space: pre-line">
Строка          1
Строка 2<br>
Строка 3
</p>
```

Результат в окне Web-браузера:

```
Строка 1
Строка 2

Строка 3
```

- `pre-wrap` — сохраняются все пробелы и переносы строк. Если текст не помещается на строке, то он будет автоматически перенесен на новую строку:

```
<p style="white-space: pre-wrap">
Строка          1
Строка 2<br>
Строка 3
</p>
```

Результат в окне Web-браузера:

```
Строка          1
Строка 2

Строка 3
```

2.5.10. Перенос слов

Управлять переносом слов позволяют следующие атрибуты:

- `hyphens` — задает способ вставки символов переноса. Чтобы атрибут правильно работал, необходимо указать язык текста в параметре `lang` (`lang="ru"`) для тега `<html>` или для абзаца. Атрибут может принимать значения `none` (символы переноса не добавляются), `manual` (символ переноса вставляется только при наличии в тексте мягких переносов ``) и `auto` (автоматическая расстановка переносов):

```
p { width: 200px; hyphens: auto }
```

Атрибут полноценно работает только в Firefox.

- `word-wrap` и `overflow-wrap` — задают способ вставки символов новой строки в длинные слова, не помещающиеся в строку по ширине. Могут принимать зна-

чения `normal` (символ новой строки автоматически не вставляется, и чтобы пометить место вставки, нужно использовать тег `<wbr>` или символ мягкого переноса ``) и `break-word` (автоматическая вставка символа новой строки — вначале длинное слово переносится на отдельную строку и только потом, если слово не помещается, вставляется символ новой строки):

```
#p1 { width: 150px; word-wrap: break-word }
...
<p id="p1">текст оченьдлинныйтекст</p>
```

Результат:

```
текст
оченьдлинныйтекст
```

- `word-break` — задает способ вставки символов новой строки в длинные слова, не помещающиеся в строку по ширине. Может принимать значения `normal` (символ новой строки автоматически не вставляется, и чтобы пометить место вставки, нужно использовать тег `<wbr>` или символ мягкого переноса ``) и `break-all` (автоматическая вставка символа новой строки):

```
#p2 { width: 150px; word-break: break-all }
...
<p id="p2">текст оченьдлинныйтекст</p>
```

Результат:

```
текст оченьдлинныйт
екст
```

2.5.11. Направление вывода текста

Направление вывода текста в CSS 3 задает атрибут `writing-mode`. Его допустимые значения:

- `horizontal-tb` — по горизонтали слева направо и сверху вниз (значение по умолчанию):

```
p { width: 200px; height: 200px; writing-mode: horizontal-tb }
```

- `vertical-rl` — по вертикали сверху вниз и справа налево:

```
p { width: 200px; height: 200px; writing-mode: vertical-rl }
```

- `vertical-lr` — по вертикали сверху вниз и слева направо:

```
p { width: 200px; height: 200px; writing-mode: vertical-lr }
```

- `sideways-lr` — по вертикали снизу вверх и слева направо (работает только в последних версиях Firefox).

2.6. Отступы

Любой элемент Web-страницы занимает в окне Web-браузера некоторую прямоугольную область. Причем эта область имеет как внутренние, так и внешние отступы. *Внутренний отступ* — это расстояние между элементом страницы и реальной

или воображаемой границей области. *Внешний отступ* — это расстояние между реальной или воображаемой границей и другим элементом Web-страницы, точнее сказать, между границей и крайней точкой внешнего отступа другого элемента Web-страницы.

По умолчанию нижний внешний отступ одного блочного элемента может объединяться с верхним внешним отступом другого блочного элемента — при этом мы получим не сумму внешних отступов, а наибольшее значение. Такой эффект можно наблюдать при использовании абзацев. Если объединение не произойдет, то отступ между абзацами увеличится в два раза. Справа и слева внешние отступы никогда не объединяются.

Чтобы увидеть структуру блочной модели, воспользуемся панелью **Инструменты разработчика** Web-браузера Firefox: открываем вкладку **Инспектор** и справа переходим на вкладку **Разметка**. Как можно видеть на рис. 2.3, вначале идут размеры элемента, затем внутренние отступы (`padding`), далее граница (`border`) и внешние отступы (`margin`).

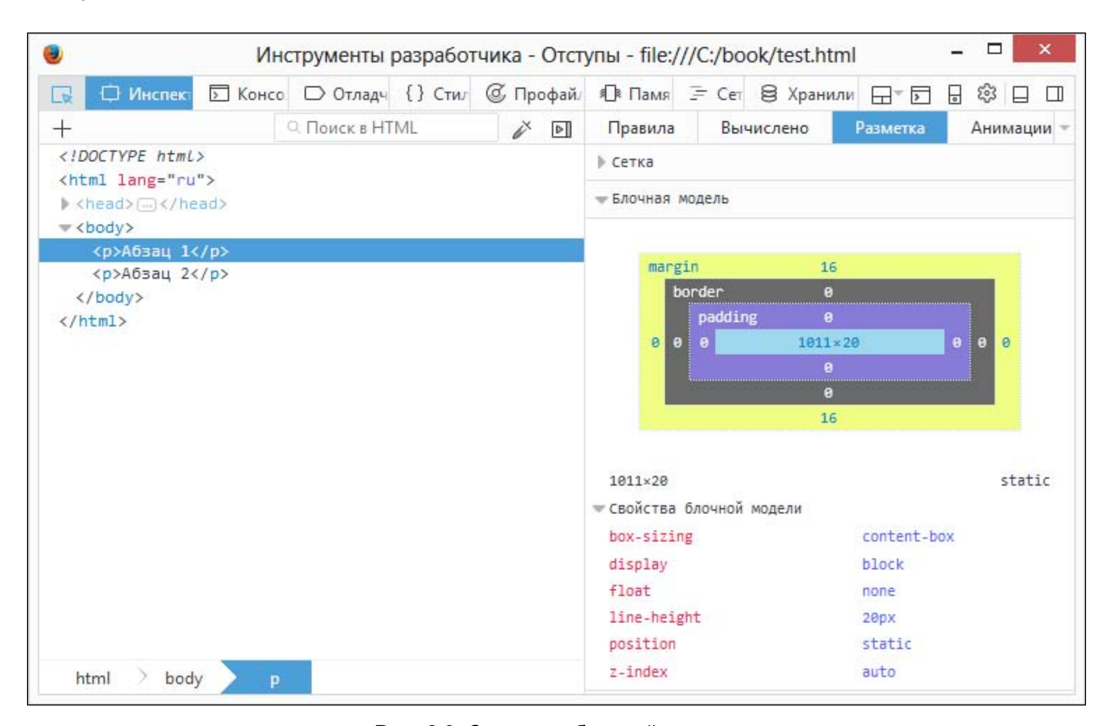


Рис. 2.3. Структура блочной модели

2.6.1. Внешние отступы

Отступы одного элемента Web-страницы от другого можно задать с помощью атрибутов `margin-left`, `margin-right`, `margin-top` и `margin-bottom`:

- `margin-left` — внешний отступ слева:

```
body { margin-left: 0 }
```

- `margin-right` — внешний отступ справа:

  ```
  body { margin-right: 5% }
  ```

- `margin-top` — внешний отступ сверху:

  ```
  body { margin-top: 15mm }
  ```

- `margin-bottom` — внешний отступ снизу:

  ```
  body { margin-bottom: 20px }
  ```

Как можно видеть, для этих атрибутов могут быть заданы абсолютные или относительные значения. Более того, атрибуты могут иметь и отрицательные значения.

Убрать все внешние отступы можно с помощью такой строки кода:

```
body { margin-left: 0; margin-right: 0; margin-top: 0; margin-bottom: 0 }
```

С помощью атрибута `margin` можно задать все внешние отступы за один раз:

```
margin: <top> <right> <bottom> <left>
margin: <top> <right и left> <bottom>
margin: <top и bottom> <right и left>
margin: <top и right и bottom и left>
```

Например:

```
body { margin: 15mm 5% 20px 0 }
```

Для совпадающих значений можно записать одно значение:

```
body { margin: 0 }
```

В качестве значения вместо конкретной величины можно указать слово `auto`, которое означает, что размер внешних отступов автоматически рассчитывается Web-браузером. Выполним горизонтальное выравнивание блоков по центру и по правой стороне:

```
div { width: 100px; height: 100px; background: green }
div.cls1 { margin: 0 auto }
div.cls2 { margin: 0 0 0 auto }
...
<div class="cls1">Текст</div>
<div class="cls2">Текст</div>
```

2.6.2. Внутренние отступы

Задать отступы от элемента Web-страницы до его рамки (если она есть) можно с помощью атрибутов `padding-left`, `padding-right`, `padding-top` и `padding-bottom`. Например, ими задается расстояние между текстом и рамкой ячейки таблицы:

- `padding-left` — внутренний отступ слева:

  ```
  td { padding-left: 0 }
  ```

- `padding-right` — внутренний отступ справа:

  ```
  td { padding-right: 50px }
  ```

- `padding-top` — внутренний отступ сверху:

  ```
  td { padding-top: 15mm }
  ```

- `padding-bottom` — внутренний отступ снизу:

  ```
  td { padding-bottom: 20px }
  ```

Как можно видеть, для этих атрибутов могут быть заданы абсолютные или относительные значения.

С помощью атрибута `padding` можно задать все внутренние отступы за один раз:

```
padding: <top> <right> <bottom> <left>
padding: <top> <right и left> <bottom>
padding: <top и bottom> <right и left>
padding: <top и right и bottom и left>
```

Например:

```
td { padding: 15mm 50px 20px 0 }
```

Совпадающие отступы можно задать и короче:

```
td { padding: 5px }
```

2.7. Рамки

Как вы уже знаете, любой элемент Web-страницы занимает в окне Web-браузера некоторую прямоугольную область. Содержимое этой области может быть окружено рамкой. Иными словами, рамки могут иметь не только таблицы, но и любые элементы Web-страницы, например абзацы.

2.7.1. Стиль линий рамки

Задать тип линий рамки можно с помощью атрибутов `border-left-style` (левая линия), `border-right-style` (правая линия), `border-top-style` (верхняя линия) и `border-bottom-style` (нижняя линия). Атрибуты могут принимать следующие значения:

- `none` — линия не отображается;
- `hidden` — если для таблицы задан атрибут `border-collapse` со значением `collapse`, то граница вокруг ячейки со значением `hidden` не будет отображаться. Для других элементов значение имеет тот же эффект, что и `none`;
- `solid` — линия отображается сплошной линией;
- `dotted` — пунктирная линия;
- `dashed` — штриховая линия;
- `double` — двойная линия;
- `groove` — вдавленная рельефная линия;
- `ridge` — выпуклая рельефная линия;

- ❒ `inset` — весь блок элемента отображается, как будто он вдавлен в лист;
- ❒ `outset` — весь блок элемента отображается, как будто он выдавлен из листа.

Эти атрибуты могут быть объединены в одном атрибуте `border-style`:

```
border-style: <top> <right> <bottom> <left>
border-style: <top> <right и left> <bottom>
border-style: <top и bottom> <right и left>
border-style: <top и right и bottom и left>
```

Если все значения совпадают, можно указать это значение один раз:

```
p { border-style: dotted }
```

В качестве примера укажем стили линий рамки для разных элементов Web-страницы (листинг 2.6).

Листинг 2.6. Стили линий рамки

```
<!DOCTYPE html>
<html lang="ru">
<head>
 <meta charset="utf-8">
 <title>Стили линий рамки</title>
 <style type="text/css">
 div, p { width: 300px; height: 100px; margin: 10px;
          border-width: 10px; padding: 5px }
 div.cls1 {
    border-top-style: solid;
    border-right-style: dotted;
    border-bottom-style: double;
    border-left-style: dashed
 }
 div.cls2 {
    border-top-style: groove;
    border-right-style: ridge;
    border-bottom-style: ridge;
    border-left-style: groove
 }
 p.cls3 { border-style: outset }
 p.cls4 { border-style: inset }
 </style>
</head>
<body>
   <div class="cls1">solid dotted double dashed</div>
   <div class="cls2">groove ridge ridge groove</div>
   <p class="cls3">outset</p>
   <p class="cls4">inset</p>
</body>
</html>
```

Этот пример показывает, что указать тип рамки можно не только для границ таблицы, но и для абзацев, а также для любых других элементов.

2.7.2. Толщина линий рамки

Задать толщину линий рамки можно с помощью атрибутов `border-left-width` (левая линия), `border-right-width` (правая линия), `border-top-width` (верхняя линия) и `border-bottom-width` (нижняя линия).

Может быть задано абсолютное значение:

```
div {
   border-style: solid;        padding: 10px;
   border-top-width: 0;        border-right-width: 5px;
   border-bottom-width: 10px; border-left-width: 15px
}
```

Также можно указать одно из предопределенных значений:

- `thin` — тонкая линия;
- `medium` — средняя толщина линии;
- `thick` — толстая линия.

```
div {
   border-style: solid;        padding: 10px;
   border-top-width: 0;        border-right-width: thick;
   border-bottom-width: thin; border-left-width: medium
}
```

Эти атрибуты могут быть объединены в одном атрибуте `border-width`:

```
border-width: <top> <right> <bottom> <left>
border-width: <top> <right и left> <bottom>
border-width: <top и bottom> <right и left>
border-width: <top и right и bottom и left>
```

Если значения совпадают, можно указать их один раз:

```
div { border-style: solid; padding: 10px; border-width: 3px }
```

2.7.3. Цвет линий рамки

Задать цвет линий рамки можно с помощью атрибутов `border-left-color` (левая линия), `border-right-color` (правая линия), `border-top-color` (верхняя линия) и `border-bottom-color` (нижняя линия):

```
div {
   border-style: solid;         padding: 10px;
   border-top-color: green;     border-right-color: #CCCCCC;
   border-bottom-color: black; border-left-color: red;
}
```

Как и в случае `border-style` и `border-width`, эти атрибуты можно объединить в один атрибут `border-color`:

```
border-color: <top> <right> <bottom> <left>
border-color: <top> <right и left> <bottom>
border-color: <top и bottom> <right и left>
border-color: <top и right и bottom и left>
```

Вот пример указания одного цвета для всех границ:

```
div { border-style: solid; padding: 10px; border-color: red }
```

2.7.4. Одновременное задание характеристик рамки

Если атрибуты рамки одинаковы для всех ее сторон, можно задавать их в одном атрибуте `border`:

```
border: <width> <style> <color>
```

Поскольку значение атрибута однозначно определяет, к какому именно компоненту он относится, то их можно указывать в произвольном порядке:

```
div { border: red thin solid }
```

Можно также указывать только отдельные значения:

```
div { border: solid red }
```

Чтобы одновременно задать характеристики рамки для одной стороны, следует воспользоваться атрибутами `border-top` (верхняя граница), `border-right` (правая граница), `border-bottom` (нижняя граница) и `border-left` (левая граница):

```
border-top:    <width> <style> <color>
border-right:  <width> <style> <color>
border-bottom: <width> <style> <color>
border-left:   <width> <style> <color>
```

Вот пример указания характеристик верхней границы:

```
div { border-top: 1px solid red }
```

2.7.5. Рамки со скругленными углами

Одна из наиболее долгожданных новых возможностей CSS 3 — скругление углов рамок и фона. Если рамка не задана, то происходит только скругление фона.

Радиус скругления углов указывается с помощью атрибутов `border-top-left-radius` (левый верхний угол), `border-top-right-radius` (правый верхний угол), `border-bottom-right-radius` (правый нижний угол) и `border-bottom-left-radius` (левый нижний угол):

```
border-top-left-radius: <Размер>[ <Размер>]
border-top-right-radius: <Размер>[ <Размер>]
border-bottom-right-radius: <Размер>[ <Размер>]
border-bottom-left-radius: <Размер>[ <Размер>]
```

Если задать два значения радиуса скругления, то первое будет применено к горизонтальной четверти угла, а второе — к вертикальной четверти. Если задано одно значение, то оно будет применено к обеим четвертям:

```
div { border: 1px solid red; height: 100px; padding: 10px;
      border-top-left-radius: 20px;
      border-top-right-radius: 20px 30px;
      border-bottom-right-radius: 20px;
      border-bottom-left-radius: 20px 30px
}
```

Указать радиус скругления сразу для всех углов рамки можно с помощью атрибута `border-radius`:

```
border-radius: <Радиус>[ / <Радиус>]
```

Перед символом слэша задается радиус скругления для горизонтальной четверти углов, а после него — радиус скругления для вертикальной четверти:

```
div { border: 1px solid red; height: 100px; padding: 10px;
      border-radius: 20px / 30px
}
```

Если после слэша значение не указать, то заданный радиус скругления будет применен и к горизонтальной, и к вертикальной четвертям:

```
div { border: 1px solid red; height: 100px; padding: 10px;
      border-radius: 20px
}
```

Параметр `<Радиус>` может быть задан следующими способами:

```
<Радиус всех четырех углов>
<top-left и bottom-right> <top-right и bottom-left>
<top-left> <top-right и bottom-left> <bottom-right>
<top-left> <top-right> <bottom-right> <bottom-left>
```

Вот пример указания всех значений:

```
div { border: 1px solid red; height: 100px; padding: 10px;
      border-radius: 10px 20px 30px 40px / 10px 20px 30px 40px
}
```

Этот пример аналогичен следующему, т. к. для обеих четвертей задано одинаковое значение:

```
div { border: 1px solid red; height: 100px; padding: 10px;
      border-radius: 10px 20px 30px 40px
}
```

2.7.6. Внешняя рамка

Помимо обычной рамки, элемент может содержать и внешнюю рамку, причем внешняя рамка не входит в состав размера элемента. Если два элемента расположе-

ны рядом, то внешняя рамка отображается поверх второго элемента, а не сдвигает его в сторону. В отличие от обычной рамки, характеристики внешней рамки мы можем задать только для всех границ сразу. Кроме того, для нее нельзя указать скругление углов.

Задать характеристики внешней рамки позволяют следующие атрибуты:

- `outline-style` — стиль линии рамки. Можно указать те же самые стили, что и для обычной рамки (за исключением `hidden`):

  ```
  p { outline-style: double }
  ```

- `outline-color` — цвет линии:

  ```
  p { outline-style: double; outline-color: red }
  ```

- `outline-width` — толщина линии. Можно указать те же самые значения, что и для обычной рамки:

  ```
  p { outline-style: double; outline-width: 10px }
  ```

- `outline-offset` — отступ от обычной границы элемента до внешней границы:

  ```
  p { outline-style: solid; outline-width: 1px;
      outline-offset: 5px }
  ```

Атрибут `outline` позволяет задать сразу несколько значений одновременно:

```
outline: <outline-color> <outline-style> <outline-width>
```

Пример:

```
div { height: 100px; width: 500px;
      padding: 10px; margin: 20px;
      border: 10px green dotted;
      outline: red double 10px;
      outline-offset: 10px
}
...
<div></div>
```

2.8. Фон элемента

Каскадные таблицы стилей позволяют задать цвет фона или использовать в качестве фона какое-либо изображение. Для фонового рисунка можно задать начальное положение и указать, будет ли рисунок прокручиваться вместе с содержимым Web-страницы. Кроме того, можно управлять повторением фонового рисунка, что позволяет получить интересные спецэффекты. Например, если в качестве фонового рисунка указать градиентную полосу с высотой, равной высоте элемента страницы, и шириной, равной 1–2 мм, а затем задать режим повторения только по горизонтали, то первоначальное изображение будет размножено по горизонтали, и мы получим градиентную полосу любой ширины.

2.8.1. Цвет фона

Задать цвет фона можно с помощью атрибута `background-color`:

```
body { background-color: green }
div { height: 100px; background-color: silver }
```

В качестве значения атрибута можно использовать слово `transparent`, которое означает, что фон прозрачный:

```
div { height: 100px; background-color: transparent }
```

2.8.2. Фоновый рисунок

Задать интернет-адрес (URL) изображения, которое будет использовано в качестве фонового рисунка, можно с помощью атрибута `background-image`. Может быть указан как абсолютный, так и относительный URL-адрес (относительно местоположения таблицы стилей, а не документа):

```
body { background-image: url(photo.jpg) }
```

Можно указать несколько изображений через запятую:

```
body { background-image: url(photo1.jpg), url(photo2.jpg) }
```

В качестве значения атрибута можно использовать слово `none`. Оно означает, что фоновая заливка отключена:

```
body { background-image: none }
```

2.8.3. Режим повторения фонового рисунка

Режим повторения фонового рисунка задает атрибут `background-repeat`. Он может принимать следующие значения:

- `repeat` — рисунок повторяется и по вертикали, и по горизонтали (по умолчанию):

  ```
  body { background-image: url(photo.jpg);
         background-repeat: repeat }
  ```

- `repeat-x` — рисунок повторяется по горизонтали:

  ```
  body { background-image: url(photo.jpg);
         background-repeat: repeat-x }
  ```

- `repeat-y` — рисунок повторяется по вертикали:

  ```
  body { background-image: url(photo.jpg);
         background-repeat: repeat-y }
  ```

- `no-repeat` — рисунок не повторяется:

  ```
  body { background-image: url(photo.jpg);
         background-repeat: no-repeat }
  ```

В CSS 3 доступны два новых значения:

- `space` — повторяет изображение без обрезки и масштабирования, при этом заполняет оставшееся место пустым пространством между изображениями:

```
body { background-image: url(photo.jpg);
       background-repeat: space }
```

- `round` — повторяет изображение таким образом, чтобы поместилось целое число изображений без пустого пространства между ними, при этом изображения масштабируются:

```
body { background-image: url(photo.jpg);
       background-repeat: round }
```

2.8.4. Прокрутка фонового рисунка

Будет ли фоновый рисунок прокручиваться вместе с документом, определяет атрибут `background-attachment`. Он может принимать следующие значения:

- `scroll` — фоновый рисунок прокручивается вместе с содержимым страницы (по умолчанию):

```
body { background-image: url(photo.jpg);
       background-repeat: no-repeat;
       background-attachment: scroll }
```

- `fixed` — фоновый рисунок не прокручивается:

```
body { background-image: url(photo.jpg);
       background-repeat: no-repeat;
       background-attachment: fixed }
```

2.8.5. Положение фонового рисунка

Начальное положение фонового рисунка задает атрибут `background-position`. В качестве значений атрибута задаются координаты в абсолютных единицах или в процентах. Координаты указываются через пробел:

```
body { background-image: url(photo.jpg); background-repeat: no-repeat;
       background-attachment: fixed; background-position: 50% 50% }
```

Кроме того, могут быть указаны следующие значения:

- `left` — выравнивание по левому краю;
- `right` — по правому краю;
- `center` — по центру;
- `top` — по верху;
- `bottom` — по низу.

Пример:

```
body { background-image: url(photo.jpg); background-repeat: no-repeat;
      background-attachment: fixed; background-position: left center }
```

2.8.6. Размеры фонового изображения

Размеры фонового изображения в CSS 3 указывает атрибут `background-size` в виде двух значений, разделенных пробелом: первое значение задает ширину изображения, второе — высоту.

Растянем фоновое изображение на всю ширину страницы и на половину ее высоты:

```
body { background-image: url(photo.jpg); background-repeat: no-repeat;
      background-size: 100% 50% }
```

Можно также указать следующие значения:

- `auto` — не изменяет размеры фонового изображения (поведение по умолчанию);
- `cover` — задает такие размеры фонового изображения, чтобы оно полностью покрывало элемент страницы. При этом какие-то фрагменты изображения могут выйти за границы элемента;
- `contain` — задает такие размеры фонового изображения, чтобы оно полностью помещалось в элементе страницы, занимая его целиком. При этом какие-то части элемента могут быть не покрыты изображением.

Заполним фоновым изображением все пространство страницы:

```
body { background-image: url(photo.jpg); background-repeat: no-repeat;
      background-size: cover }
```

2.8.7. Режим позиционирования фонового изображения

Если мы задали местоположение фонового изображения, то можем также указать, относительно чего оно будет позиционироваться. Для этого в CSS 3 мы применим атрибут `background-origin`, задав одно из следующих его значений:

- `padding-box` — изображение будет позиционироваться относительно границ внутренних отступов (поведение по умолчанию);
- `border-box` — позиционирование фонового изображения будет выполняться относительно рамки элемента;
- `content-box` — изображение будет позиционироваться относительно границ содержимого элемента.

Позиционируем фоновое изображение относительно содержимого контейнера с идентификатором `main`:

```
#main { padding: 10px; border: 10px red dashed;
       background-image: url(photo.jpg);
```

```
        background-position: center top;
        background-repeat: no-repeat;
        background-origin: border-box }
```

2.8.8. Режим заполнения для фона

Иногда полезно указать, какую часть элемента будет заполнять фон. За это в CSS 3 отвечает атрибут `background-clip`, поддерживающий три значения:

- `content-box` — фоном будет заполнена часть элемента, занятая его содержимым;
- `padding-box` — фоном будет заполнена часть элемента, занятая содержимым и внутренними отступами;
- `border-box` — фоном будет заполнен весь элемент (поведение по умолчанию).

Заполним черным цветом лишь ту часть контейнера `main`, что занята собственно содержимым:

```
#main { padding: 10px; border: 10px red dashed;
        color: white; background-color: black;
        background-clip: content-box }
```

2.8.9. Одновременное задание характеристик фона

Атрибут `background` является сокращенным вариантом для одновременного указания значений атрибутов `background-color`, `background-image`, `background-position`, `background-repeat`, `background-attachment` и др.:

```
body { background: green url(photo.jpg) no-repeat fixed left center }
div { background: green }
```

Обратите внимание на то, что во втором примере мы указали только цвет фона. Если остальные атрибуты не указаны, то они получают значения по умолчанию. Кроме того, поскольку значение атрибута в большинстве случаев однозначно определяет, к какому именно компоненту он относится, то их можно указывать в произвольном порядке.

2.9. Градиентные фоны

В CSS 3, помимо заливки сплошным цветом и фоновым рисунком, существует возможность выполнить заливку фона линейным или радиальным градиентом. Градиентный фон указывают либо с помощью атрибута `background-image`, либо в составе атрибута `background`.

2.9.1. Линейные градиенты

В линейном градиенте переходящие друг в друга цвета распространяются по прямой из одной точки в другую. Точка, в которой градиент начинается, называется *начальной*, а точка, в которой он заканчивается, — *конечной*.

Линейный градиент формируется с помощью функции `linear-gradient()`:

```
linear-gradient([ [<Угол>][ to <Направление>] ,]
         <Цвет 1>[ <Положение 1>], ..., <Цвет N>[ <Положение N>])
```

Любой градиент должен иметь, по крайней мере, две *ключевые точки*, которые часто располагаются в его начальной и конечной точках. Если указаны только два цвета, то градиент станет распространяться по вертикали сверху вниз: первый цвет будет находиться в начальной точке, а второй — в конечной.

Вот пример плавного перехода по вертикали от красного цвета до синего:

```
#main { height: 200px; border: 1px red solid;
        background-image: linear-gradient(red, blue)
}
...
<div id="main"></div>
```

Ключевая точка располагается на воображаемой прямой, проведенной между начальной и конечной точками градиента, и определяет, какой «чистый» цвет должен присутствовать в этом месте. При этом цвета, которые будут присутствовать между ключевыми точками, сформируются переходом одного «чистого» цвета в другой.

Ключевая точка характеризуется расстоянием между ней и начальной точкой градиента и, собственно, значением цвета. Положение ключевой точки обычно указывают в процентах, чтобы связать размеры градиента с размерами элемента, фоном которого он станет. Предыдущий пример мы можем записать следующим образом:

```
background-image: linear-gradient(red 0%, blue 100%)
```

Если положение первой ключевой точки градиента не указано, она будет находиться в начальной точке (`0%`). Если не указать положение последней ключевой точки, она будет находиться в конечной точке (`100%`). А если не указывать положения промежуточных ключевых точек, они выстроятся по градиенту на одинаковом расстоянии друг от друга:

```
background-image: linear-gradient(red, green, blue)
```

Эта запись эквивалентна такой:

```
background-image: linear-gradient(red 0%, green 50%, blue 100%)
```

Если для первой точки указано значение, отличное от `0%`, то пространство от начальной точки до первой точки будет залито сплошным цветом первой точки. Если для последней точки указано значение, отличное от `100%`, то пространство от последней точки до конечной точки будет залито сплошным цветом последней точки:

```
background-image: linear-gradient(red 25%, green 75%)
```

В этом примере пространство от `0%` до `25%` будет залито сплошным красным цветом, от `25%` до `75%` — залито плавным градиентом от красного цвета до зеленого, а пространство от `75%` до `100%` — залито сплошным зеленым цветом.

Направление, по которому будет распространяться градиент, можно указать следующими способами:

- в виде значения угла между горизонталью и линией, по которой должен распространяться градиент. Этот угол отсчитывается по часовой стрелке. Для указания значения угла CSS 3 предлагает следующие единицы измерения: `deg` (градусы), `rad` (радианы), `grad` (грады) и `turn` (повороты).

  Вот пример градиента по горизонтали от черного цвета (слева) до белого (справа):

  ```
  background-image: linear-gradient(90deg, black, white)
  ```

  Если требуется отсчитывать угол против часовой стрелки, нужно указать отрицательное значение. Вот пример градиента по горизонтали от черного (справа) до белого (слева):

  ```
  background-image: linear-gradient(-90deg, black, white)
  ```

- с помощью одного из значений (или их комбинации через пробел), указанных после ключевого слова `to`: `top` (вверх), `bottom` (вниз), `left` (налево) и `right` (направо). В этом случае градиент будет распространяться в указанном направлении по горизонтали или вертикали.

  Вот пример градиента по вертикали снизу вверх:

  ```
  background-image: linear-gradient(to top, black, white)
  ```

  Если направление не указано, градиент будет распространяться сверху вниз (как будто для него было задано значение направления `bottom`):

  ```
  background-image: linear-gradient(to bottom, black, white)
  ```

  Пример градиента по горизонтали слева направо:

  ```
  background-image: linear-gradient(to right, black, white)
  ```

  Пример градиента по горизонтали справа налево:

  ```
  background-image: linear-gradient(to left, black, white)
  ```

  Можно указать комбинацию из двух значений, разделенных пробелом. В этом случае градиент будет распространяться по диагонали в угол, образованный смыканием указанных сторон. Вот пример градиента по диагонали от правого нижнего угла до левого верхнего угла:

  ```
  background-image: linear-gradient(to top left, black, white)
  ```

Повторяющийся линейный градиент можно создать с помощью функции `repeating-linear-gradient()`:

```
repeating-linear-gradient([ [<Угол>][ to <Направление>] ,]
          <Цвет 1>[ <Положение 1>], ..., <Цвет N>[ <Положение N>])
```

Все параметры аналогичны одноименным параметрам функции `linear-gradient()`, но последняя точка будет конечной точкой градиента.

Пример повторяющегося линейного красно-синего градиента:

```
background-image: repeating-linear-gradient(red, blue 100px)
```

Пример чередования красных и белых полос под углом 45 градусов:

```
background-image: repeating-linear-gradient(
          45deg, red, red 20px, white 20px, white 40px)
```

2.9.2. Радиальные градиенты

В радиальном градиенте переходящие друг в друга цвета распространяются из начальной точки во все стороны, в конечном счете образуя своего рода эллипс (или круг). За конечную точку такого градиента принимается любая точка, расположенная на охватывающем его воображаемом эллипсе.

Радиальный градиент описывается функцией `radial-gradient()` по следующей схеме:

```
radial-gradient( [[ [<Форма> ][<Радиус> ][<Размер> ] ]
          [ at <Положение начальной точки>],]
          <Цвет 1>[ <Положение 1>], ..., <Цвет N>[ <Положение N>])
```

Вот пример радиального градиента в форме эллипса от красного цвета (в центре элемента) до синего (по краям):

```
#main { height: 200px; border: 1px red solid;
        background-image: radial-gradient(red, blue)
}
...
<div id="main"></div>
```

Положение начальной точки радиального градиента (положение центра эллипса или круга) задается в том же формате, что и значение атрибута `background-position` (см. *разд. 2.8.5*). Если параметр `<Положение начальной точки>` не указан, то начальная точка будет располагаться в центре элемента.

Пример смещения начальной точки в левый верхний угол:

```
background-image: radial-gradient(at left top, red, blue)
```

Набор ключевых точек задается точно так же, как и у линейного градиента:

```
background-image: radial-gradient(at 0 50%,
                                  red 0%, green 50%, blue 100%)
```

Параметр `<Форма>` может принимать два значения:

- `ellipse` — градиент эллиптической формы (значение по умолчанию):

  ```
  background-image: radial-gradient(ellipse, red, blue)
  ```

  По умолчанию эллипс, на основе которого формируется градиент, будет иметь то же соотношение сторон, что и элемент, фоном которого он станет. С помощью параметра `<Радиус>` можно указать другое соотношение сторон:

  ```
  background-image: radial-gradient(ellipse 150% 50%, red, blue)
  ```

  Первое значение задает радиус эллипса по оси `X`, а второй — по оси `Y`. Ключевое слово `ellipse` можно не указывать:

  ```
  background-image: radial-gradient(150% 50%, red, blue)
  ```

- `circle` — градиент в форме круга:

  ```
  background-image: radial-gradient(circle at left top, red, blue)
  ```

  Параметр `<Радиус>` позволяет задать радиус круга:

  ```
  background-image: radial-gradient(circle 100px, red, blue)
  ```

  Ключевое слово `circle` в этом случае можно не указывать:

  ```
  background-image: radial-gradient(100px, red, blue)
  ```

В параметре `<Размер>` можно указать размер градиента в виде одного из следующих значений:

- `closest-side` — градиент будет заканчиваться у самой близкой к его начальной точке стороны элемента страницы:

  ```
  background-image: radial-gradient(closest-side at 100px 30px,
                                    white, blue)
  ```

- `closest-corner` — градиент будет заканчиваться у самого близкого к его начальной точке угла элемента страницы:

  ```
  background-image: radial-gradient(closest-corner at 100px 30px,
                                    white, blue)
  ```

- `farthest-side` — градиент будет заканчиваться у самой дальней от его начальной точки стороны элемента страницы:

  ```
  background-image: radial-gradient(farthest-side at 100px 30px,
                                    white, blue)
  ```

- `farthest-corner` — градиент будет заканчиваться у самого дальнего от его начальной точки угла элемента страницы (значение по умолчанию):

  ```
  background-image: radial-gradient(farthest-corner at 100px 30px,
                                    white, blue)
  ```

  Для градиента круглой формы нужно дополнительно указать слово `circle`:

  ```
  background-image: radial-gradient(
      circle farthest-corner at 100px 30px, white, blue)
  ```

Повторяющийся радиальный градиент можно создать с помощью функции `repeating-radial-gradient()`:

```
repeating-radial-gradient([[ [<Форма> ][<Радиус> ][<Размер> ] ]
          [ at <Положение начальной точки>],]
          <Цвет 1>[ <Положение 1>], ..., <Цвет N>[ <Положение N>])
```

Все параметры аналогичны одноименным параметрам функции `radial-gradient()`, но последняя точка будет конечной точкой градиента.

Пример повторяющегося круглого радиального бело-синего градиента:

```
background-image: repeating-radial-gradient(
           circle farthest-corner at 80px 40px, white, blue 20px)
```

Пример чередования красных и белых полос:

```
background-image: repeating-radial-gradient(
     circle at 80px 40px, red, red 20px, white 20px, white 40px)
```

2.10. Списки

Задать параметры списка можно с помощью атрибутов стиля. Мы можем указать вид маркера, отображаемого перед пунктами, а также его положение. Кроме того, каскадные таблицы стилей позволяют использовать в качестве маркера списка любое изображение.

2.10.1. Вид маркера списка

Вид маркера списка задает атрибут `list-style-type`. Он может принимать следующие значения:

- `disc` — выводит значки в форме кружков с заливкой:

  ```
  ul { list-style-type: disc }
  ```

- `circle` — выводит значки в форме кружков без заливки:

  ```
  ul { list-style-type: circle }
  ```

- `square` — выводит значки в форме квадрата с заливкой:

  ```
  ul { list-style-type: square }
  ```

- `decimal` — нумерует пункты арабскими цифрами:

  ```
  ol { list-style-type: decimal }
  ```

- `decimal-leading-zero` — нумерует пункты арабскими цифрами. Для чисел меньше десяти дополнительно выводится ведущий нуль (`01`):

  ```
  ol { list-style-type: decimal-leading-zero }
  ```

- `lower-roman` — нумерует пункты малыми римскими цифрами:

  ```
  ol { list-style-type: lower-roman }
  ```

- `upper-roman` — нумерует пункты большими римскими цифрами:

  ```
  ol { list-style-type: upper-roman }
  ```

- `lower-alpha` и `lower-latin` — нумеруют пункты строчными латинскими буквами:

  ```
  ol { list-style-type: lower-alpha }
  ol { list-style-type: lower-latin }
  ```

- `upper-alpha` и `upper-latin` — нумеруют пункты прописными латинскими буквами:

  ```
  ol { list-style-type: upper-alpha }
  ol { list-style-type: upper-latin }
  ```

- `lower-greek` — нумерует пункты малыми греческими буквами:

  ```
  ol { list-style-type: lower-greek }
  ```

- `georgian` — нумерует пункты грузинскими буквами:

  ```
  ol { list-style-type: georgian }
  ```

- `armenian` — нумерует пункты армянскими буквами:

  ```
  ol { list-style-type: armenian }
  ```

- `none` — никак не помечает пункты списка:

  ```
  ol { list-style-type: none }
  ```

С помощью псевдоэлемента `::before` в качестве маркера списка можно указать любой символ Unicode:

```
ul { list-style-type: none; padding-left: 20px }
ul li::before {
   content: "\06DE";
   color: red;
   margin-right: 0.5em
}
```

Заметим также, что атрибут `list-style-type` (а также остальные атрибуты списков) можно применять не только к списками, но и к другим элементам, у которых атрибут `display` имеет значение `list-item`:

```
div { display: list-item; list-style-type: circle; margin-left: 40px }
```

2.10.2. Изображение в качестве маркера списка

URL-адрес изображения, которое будет использовано в качестве маркера списка, задает атрибут `list-style-image`. При этом относительные адреса указываются относительно расположения таблицы стилей, а не HTML-документа:

```
ul { list-style-image: url(marker.png) }
```

Значение `none` отменяет использование изображения в качестве маркера списка:

```
ul { list-style-image: none }
```

2.10.3. Компактное отображение списка

Более компактное отображение пунктов списка позволяет задать атрибут `list-style-position`. Он может принимать следующие значения:

- `outside` — по умолчанию. Маркер отображается отдельно от текста:

  ```
  ol { list-style-position: outside }
  ```

- `inside` — более компактное отображение списка. Маркер входит в состав текста:

  ```
  ol { list-style-position: inside }
  ```

2.10.4. Одновременное указание характеристик списка

Указать за один раз все характеристики списка позволяет атрибут `list-style`:

```
list-style: <type> <position> <image>
```

Параметры не являются обязательными, поэтому некоторые можно не указывать:

```
ul { list-style: circle inside }
ul { list-style: url(marker.png) }
```

2.11. Таблицы

В HTML 5 все оформление таблиц выполняется с помощью CSS. В листинге 1.11 мы уже рассмотрели пример такого оформления. В этом разделе мы более подробно остановимся на нескольких моментах.

2.11.1. Рамки таблицы и ячеек

По умолчанию таблица выводится без рамки. Чтобы добавить рамку, нужно воспользоваться атрибутом `border`. Причем его можно указать как для таблицы, так и для ячеек:

```
table {
   border: black 1px solid;     /* Стиль рамки таблицы */
}
td, th {
   border: black 1px solid;     /* Стиль рамки ячеек */
}
```

По умолчанию между ячейками имеется некоторое расстояние, поэтому рамка каждой ячейки отображается отдельно. Управлять этим расстоянием позволяет атрибут `border-spacing`:

```
border-spacing: 0;          /* Расстояние между ячейками */
```

Можно указать сразу два значения через пробел. В этом случае первое значение определяет расстояние по горизонтали, а второе — по вертикали:

```
table { border-spacing: 0 5px }
```

Атрибут `border-collapse` позволяет установить режим рисования рамок для самой таблицы и для ее ячеек. Поддерживаются два значения:

- `separate` — рисуется рамка вокруг самой таблицы и рамки вокруг каждой из ее ячеек (поведение по умолчанию):

  ```
  table { border-collapse: separate }
  ```

- `collapse` — рисуются лишь рамки, разделяющие ячейки. Каждая ячейка таблицы будет заключена в одинарную рамку. Значение атрибута `border-spacing` при этом игнорируется:

  ```
  table { border-collapse: collapse }
  ```

2.11.2. Размеры таблицы

Указать ширину таблицы или ячеек позволяет атрибут `width`:

```
table { width: 300px }
```

Задать высоту таблицы или ячеек можно с помощью атрибута `height`:

```
td, th { width: 150px; height: 50px }
```

Режим задания размеров у таблиц определяет атрибут `table-layout`:

- `auto` — заданные нами значения ширины таблицы и ее ячеек — это лишь рекомендация для Web-браузера (значение по умолчанию). Он может изменить ширину таблицы и ее ячеек, если содержимое в них не помещается:

  ```
  table { width: 300px; table-layout: auto }
  ```

- `fixed` — ширина таблицы и ее ячеек ни в коем случае изменяться не будет. Если содержимое не помещается в ячейках, возникнет переполнение, параметры которого мы можем задавать с помощью атрибута `overflow`:

  ```
  table { width: 300px; overflow: auto; table-layout: fixed }
  ```

  Теперь ширина таблицы не будет изменяться ни в коем случае.

Если нужно выполнить горизонтальное выравнивание таблицы по центру родительского элемента, то можно воспользоваться атрибутом `margin`:

```
table { width: 300px; margin: 0 auto }
```

Вот пример выравнивания по правой стороне:

```
table { width: 300px; margin: 0 0 0 auto }
```

2.11.3. Местоположение заголовка

Задать местоположение заголовка таблицы позволяет атрибут `caption-side`:

- `top` — заголовок отображается над таблицей:

  ```
  caption {
     text-align: center; /* Выравнивание заголовка по центру */
     caption-side: top   /* Вывод заголовка над таблицей */
  }
  ```

- `bottom` — заголовок отображается под таблицей:

  ```
  caption {
     caption-side: bottom /* Вывод заголовка под таблицей */
  }
  ```

2.11.4. Указание характеристик ячеек

Выполнить выравнивание содержимого ячейки по горизонтали позволяет атрибут `text-align` (см. *разд. 2.5.5*), а по вертикали — атрибут `vertical-align`

(см. *разд. 2.5.6*). Задать величину отступа между границей ячейки и ее содержимым можно с помощью атрибута `padding` (см. *разд. 2.6.2*):

```
td, th {
   height: 50px;
   border: black 1px solid;      /* Стиль рамки ячеек */
   padding: 10px;            /* Отступ между границей и содержимым */
   text-align: center;           /* Выравнивание ячейки по центру */
   vertical-align: top           /* Выравнивание ячейки по верху */
}
```

Изменить цвет фона ячеек таблицы можно с помощью атрибутов `background-color` и `background`. Чтобы читать таблицу было удобнее, для четных и нечетных строк задают разные цвета фона, например, с помощью псевдокласса `:nth-child`:

```
tr:nth-child(2n) {
   background: #e8e8e8          /* Зебра */
}
```

Вид ячейки без содержимого задает атрибут `empty-cells` (если ячейка содержит пробел, символ табуляции или символ перевода строки, то она также считается пустой):

- `show` — границы и фон пустой ячейки отображаются:

  ```
  td { border: red 1px solid; background: gray; empty-cells: show }
  ```

- `hide` — границы и фон пустой ячейки не отображаются:

  ```
  td { border: red 1px solid; background: gray; empty-cells: hide }
  ```

Чтобы не было проблем с Web-браузерами, лучше в пустую ячейку вставлять неразрывный пробел ` `:

```
<td> </td>
```

2.12. Вид курсора

Форму курсора мыши при наведении ее на элемент страницы задает атрибут `cursor`. Он может принимать следующие значения:

- `auto` — Web-браузер сам определяет форму курсора мыши:

  ```
  p { cursor: auto }
  ```

- `none` — курсор не отображается:

  ```
  p { cursor: none }
  ```

- `crosshair` — в виде креста:

  ```
  p { cursor: crosshair }
  ```

- `default` — стрелка (курсор по умолчанию):

  ```
  p { cursor: default }
  ```

- `pointer` — в виде руки с указывающим пальцем:

  ```
  p { cursor: pointer }
  ```

- `move` и `all-scroll` — стрелки, указывающие во всех направлениях:

  ```
  p { cursor: move }
  ```

- `n-resize`, `ne-resize`, `nw-resize`, `s-resize`, `se-resize`, `sw-resize`, `e-resize`, `w-resize`, `nesw-resize`, `nwse-resize`, `col-resize`, `row-resize` — стрелки, показывающие направление:

  ```
  p { cursor: n-resize }
  ```

- `text` и `vertical-text` — текстовый курсор:

  ```
  p { cursor: text }
  ```

- `wait` — курсор ожидания (песочные часы или круговой индикатор):

  ```
  p { cursor: wait }
  ```

- `progress` — стрелка с песочными часами или круговым индикатором:

  ```
  p { cursor: progress }
  ```

- `help` — стрелка с вопросительным знаком:

  ```
  p { cursor: help }
  ```

- `cell` — курсор ячейки таблицы:

  ```
  p { cursor: cell }
  ```

- `copy` — курсор копирования (стрелка с плюсом):

  ```
  p { cursor: copy }
  ```

- `not-allowed` и `no-drop` — курсор в виде перечеркнутого круга:

  ```
  p { cursor: not-allowed }
  ```

Прочие курсоры: `alias`, `zoom-in`, `zoom-out`, `grab`, `grabbing` и `context-menu`.

Можно также задать свой собственный курсор:

```
p { cursor: url(mycursor.png), url(mycursor.cur), pointer }
```

После адреса файла с изображением курсора допускается указание координат горячей точки:

```
p { cursor: url(mycursor.png) 2 2, pointer }
```

2.13. Псевдостили гиперссылок. Отображение ссылок разными цветами

Web-браузеры позволяют отобразить посещенные и непосещенные ссылки разными цветами. Достигается это при помощи следующих псевдоклассов:

- `:link` — вид непосещенной ссылки;
- `:visited` — вид посещенной ссылки;

- `:active` — вид активной ссылки;
- `:hover` — вид ссылки, на которую указывает курсор мыши.

> **Внимание!**
> До и после двоеточия не должно быть пробелов.

Пример:

```
a:link { color: #000000 }
a:visited { color: #000080 }
a:active { color: #FF0000 }
...
<a href="doc1.html">Ссылка1</a>
```

С помощью псевдостилей можно менять не только цвет ссылки, но и задать, будет ли ссылка подчеркнута:

```
a:link { color: red; text-decoration: underline }
a:visited { color: blue; text-decoration: underline }
a:active { color: green; text-decoration: none }
a:hover { color: lime; text-decoration: none }
```

Кроме того, можно применить стиль гиперссылок не только ко всему документу, но и к определенному классу:

```
a.link1:link { color: black; text-decoration: none }
a.link1:visited { color: blue; text-decoration: none }
a.link1:active { color: red; text-decoration: underline }
a.link1:hover { color: red; text-decoration: underline }
...
<a href="doc2.html" class="link1">Ссылка2</a>
```

2.14. Форматирование блоков

Как вы уже знаете, любой элемент Web-страницы занимает в окне Web-браузера некоторую прямоугольную область. Эта область имеет как внутренние, так и внешние отступы, а также содержит реальную или воображаемую границу. Тип блочной модели зависит от формата документа. Если тег `<!DOCTYPE>` указан, то блочная модель соответствует стандартам консорциума W3C. Реальные размеры элемента вычисляются так (см. рис. 2.3):

```
Реальная ширина = margin-left + border-left-width + padding-left +
                  width + padding-right + border-right-width +
                  margin-right
Реальная высота = margin-top + border-top-width + padding-top + height +
                  padding-bottom + border-bottom-width + margin-bottom
```

Если тег `<!DOCTYPE>` не указан, то некоторые Web-браузеры переходят в режим совместимости. Поэтому при отсутствии тега `<!DOCTYPE>` разные Web-браузеры могут по-разному отображать Web-страницу.

2.14.1. Указание типа блока

Для указания типа блока предназначен атрибут `display`. Он может принимать следующие значения:

- `none` — содержимое блока не отображается;
- `block` — блок занимает всю ширину родительского элемента. Значение `block` по умолчанию имеют блочные теги `<div>` и `<p>`;
- `inline` — блок занимает только необходимое для отображения содержимого пространство. Задать размеры блока нельзя. Значение `inline` по умолчанию имеют строчные теги `<span>`, `<b>` и др.;
- `inline-block` — аналогично `inline`, но дополнительно можно задать размеры и другое форматирование, применяемое для блочного элемента. Результат аналогичен встраиванию тега `<img>` в строку;
- `list-item` — пункт списка. Это значение по умолчанию имеет тег `<li>`;
- `table` — блочная таблица. Это значение по умолчанию имеет тег `<table>`;
- `inline-table` — строчная таблица;
- `table-caption` — заголовок таблицы. Это значение по умолчанию имеет тег `<caption>`;
- `table-header-group` — это значение по умолчанию имеет тег `<thead>`;
- `table-row-group` — это значение по умолчанию имеет тег `<tbody>`;
- `table-footer-group` — это значение по умолчанию имеет тег `<tfoot>`;
- `table-row` — строка таблицы. Это значение по умолчанию имеет тег `<tr>`;
- `table-column` — это значение по умолчанию имеет тег `<col>`;
- `table-column-group` — это значение по умолчанию имеет тег `<colgroup>`;
- `table-cell` — ячейка таблицы. Это значение по умолчанию имеют теги `<th>` и `<td>`;
- `flex` — блочный flex-контейнер (см. *разд. 2.16*);
- `inline-flex` — строчный flex-контейнер;
- `grid` — блочный grid-контейнер (см. *разд. 2.17*);
- `inline-grid` — строчный grid-контейнер;
- `subgrid` — вложенный grid-контейнер.

Различные варианты использования атрибута `display` приведены в листинге 2.7.

Листинг 2.7. Атрибут `display`

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
```

```
    <title>Атрибут display</title>
    <style type="text/css">
       div div { border: 2px solid #333 }
       label { display: inline-block; width: 100px }
    </style>
</head>
<body>
 <h1>Различные типы блоков</h1>
 <div>
    <div style="display: inline">display = inline</div>
    <div style="display: inline-block; width: 300px">
      display = inline-block
    </div>
    <div style="display: block">display = block</div>
    <div style="display: none">Этого блока не видно</div>
 </div>

 <h2>Выравнивание элементов формы</h2>
 <div>
    <label for="login">Логин:</label>
    <input type="text" name="login" id="login">
 </div>
 <div>
    <label for="passw">Пароль:</label>
    <input type="password" name="passw" id="passw">
 </div>

 <h2>Указание типа блока для ссылки</h2>
 <div>
    <div style="width: 300px">
       <a href="link1.html">Обычная ссылка</a>
    </div>
    <div style="width: 300px">
      <a href="link.html" style="display: block">Ссылка занимает всю
                                                 ширину блока</a>
    </div>
 </div>
</body>
</html>
```

2.14.2. Указание размеров блока

Определить режим, в котором будут задаваться размеры блока, позволяет атрибут `box-sizing`. Он может принимать следующие два значения:

- `content-box` — размеры будут задаваться для содержимого блока без учета его внутренних отступов и рамки (это поведение по умолчанию):

```
#main { width: 400px; padding: 5px; box-sizing: content-box }
```

Элемент с идентификатором `main` получит ширину 410 пикселов, а его содержимое будет иметь ширину 400 пикселов;

- `border-box` — размеры будут задаваться для самого блока с учетом его внутренних отступов и рамки (без учета внешних отступов):

```
#main { width: 400px; padding: 5px; box-sizing: border-box }
```

Элемент с идентификатором `main` получит ширину 400 пикселов, а его содержимое будет иметь ширину 390 (400 – 2×5) пикселов.

Указать размеры позволяют следующие атрибуты:

- `width` и `height` — задают соответственно ширину и высоту блока (значение по умолчанию `auto`):

```
div { width: 100px; height: 100px; background: green }
```

- `min-width` и `min-height` — задают соответственно минимальные ширину и высоту блока:

```
div { min-width: 100px; min-height: 100px; background: green }
```

- `max-width` и `max-height` — задают соответственно максимальную ширину и высоту блока:

```
div { max-width: 100px; max-height: 100px; background: green }
```

Эти атрибуты могут иметь значение `none`, означающее, что значение не задано.

По умолчанию атрибуты `width` и `height` имеют значение `auto`. Если размеры явно не заданы, то блок займет всю ширину родительского элемента, а его высота будет определена по содержимому блока. Если содержимое не помещается в блок заданного размера, то он будет отображаться в соответствии со значением атрибута `overflow` (см. далее).

> ***Примечание***
> Указать размеры для строчных элементов нельзя. Строчным элементам нужно предварительно задать значение `inline-block` атрибута `display`.

2.14.3. Атрибут *overflow*

Поведение блока, чье содержимое выходит за его границы, задает атрибут `overflow`. Он может принимать следующие значения:

- `visible` — блок увеличивается в размерах так, чтобы все его содержимое отобразилось полностью (значение по умолчанию). Если размеры заданы явным образом, то содержимое будет выходить за границы блока, а размеры самого блока останутся прежними;
- `hidden` — то, что не помещается в блоке, скрывается;
- `scroll` — у блока в любом случае отображаются полосы прокрутки;

- `auto` — если содержимое не помещается в блок, то добавляются полосы прокрутки. Если же содержимое полностью помещается, то полосы прокрутки не отображаются.

С помощью атрибутов `overflow-x` и `overflow-y` можно задать разные значения для горизонтального и вертикального направления соответственно:

```
div { overflow-x: hidden; overflow-y: auto }
```

Различные варианты использования атрибута `overflow` приведены в листинге 2.8.

Листинг 2.8. Атрибут `overflow`

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Атрибут overflow</title>
   <style type="text/css">
      body { font-size: 14px; font-family: Arial; color: black }
      .div1 div { width: 100px; height: 100px }
      .div1 div, .div2 div {
          background-color: silver;
          border: black 2px solid;
          margin-bottom: 10px
      }
      .div2 { height: 600px }
      p { font-weight: bold }
   </style>
</head>
<body>
 <div class="div1">
  <p>overflow = hidden</p>
  <div style="overflow: hidden">
  этооченьдлиннаястрокабезпробелов
  То, что не влезает в границы блока, скрывается
  </div>
  <p>overflow = scroll</p>
  <div style="overflow: scroll">
  этооченьдлиннаястрокабезпробелов
  overflow = scroll. У блока в любом случае отображаются полосы прокрутки
  </div>
  <p>overflow = auto</p>
  <div style="overflow: auto">
  overflow = auto
  </div>
  <div style="overflow: auto">
```

```
  этоочень длиннаястрокабезпробелов
  overflow = auto. Если содержимое не помещается в блок, то добавляются
  полосы прокрутки
  </div>
 </div>
 <div class="div2">
  <p>overflow = visible. Высота не задана</p>
  <div style="overflow: visible; width: 100px">
  этоочень длиннаястрокабезпробелов
  overflow = visible. Блок расширяется так, чтобы все его содержимое
  отобразилось полностью
  </div>
  <p>overflow = visible. Высота задана</p>
  <div style="overflow: visible; width: 100px; height: 100px">
  этоочень длиннаястрокабезпробелов
  overflow = visible. Если размеры заданы, то содержимое будет выходить
  за границы блока
  </div>
 </div>
</body>
</html>
```

Если атрибут `overflow` имеет значение `scroll`, `hidden` или `auto` (но не `visible`), то с помощью атрибута `resize` можно указать способ изменения размеров блока мышью. Атрибут может принимать следующие значения:

- `none` — размеры блока изменить нельзя;
- `horizontal` — можно только по горизонтали;
- `vertical` — можно только по вертикали;
- `both` — размеры блока можно менять и по горизонтали, и по вертикали.

Пример:

```
<div style="width: 200px; height: 100px; overflow: scroll; resize: both">
  У блока в любом случае отображаются полосы прокрутки.
  Размеры можно менять с помощью мыши, взявшись за маркер
  в правом нижнем углу
</div>
```

Если атрибут `overflow` имеет значение `scroll`, `hidden` или `auto` (но не `visible`), то с помощью атрибута `text-overflow` можно указать способ обрезки текста, если он не помещается в область. Атрибут может принимать следующие значения:

- `clip` — текст просто обрезается (значение по умолчанию);
- `ellipsis` — текст обрезается и в конец добавляется многоточие:

```
#p1 { width: 200px; white-space: nowrap;
      overflow: hidden; text-overflow: ellipsis }
...
<p id="p1">Очень длинный текст, который будет обрезан</p>
```

Результат:

```
Очень длинный текст, кото...
```

2.14.4. Управление обтеканием

По какой стороне производится выравнивание блока, определяет атрибут `float`. Он может принимать следующие значения:

- `left` — блок выравнивается по левой стороне, а другие элементы обтекают его справа;
- `right` — блок выравнивается по правой стороне, а другие элементы обтекают его слева;
- `none` — выравнивание отсутствует (значение по умолчанию).

Разрешает или запрещает обтекание атрибут `clear`. Он может принимать следующие значения:

- `both` — запрещает обтекание по обеим сторонам;
- `left` — запрещает обтекание по левой стороне;
- `right` — запрещает обтекание по правой стороне;
- `none` — отменяет запрет на обтекание, который был установлен с помощью значений `both`, `left` и `right`.

Очень часто атрибуты `float` и `clear` применяются в блочной верстке Web-страницы для создания колонок. Рассмотрим это на примере (листинг 2.9).

Листинг 2.9. Блочная верстка страницы с помощью атрибута `float`

```
<!DOCTYPE html>
<html lang="ru">
<head>
 <meta charset="utf-8">
 <title>Блочная верстка страницы с помощью float</title>
 <style type="text/css">
  * { margin: 0; padding: 0 } /* Убираем все отступы */
  body { font-family: Verdana, Tahoma, sans-serif; font-size: 14px }
  header { background-color: silver; padding: 10px; height: 50px;
           text-align: center; line-height: 50px }
  nav   { float: left; border: 1px solid black; width: 150px;
          padding: 5px; margin: 10px; min-height: 200px }
  section { border: 1px solid black; padding: 5px;
          margin: 10px 10px 10px 185px; min-height: 500px }
  footer { background-color: silver; padding: 5px; clear: both;
           height: 30px; line-height: 30px }
 </style>
</head>
```

```
<body>
 <header><h1>Заголовок</h1></header>
 <nav>Панель навигации
    <p><a href="page2.html">Страница 2</a></p>
    <p><a href="page3.html">Страница 3</a></p>
 </nav>
 <section>
    <h2>Основное содержимое страницы</h2>
    <p>Какой-то текст</p>
 </section>
 <footer>Информация об авторских правах</footer>
</body>
</html>
```

Создаваемая Web-страница состоит из четырех блоков. Первый блок (`header`) содержит заголовок и занимает всю ширину окна. Второй блок (`nav`) предназначен для вывода панели навигации — для этого блока указано выравнивание по левой стороне и обтекание справа. В третьем блоке (`section`) располагается основное содержимое Web-страницы — этот блок занимает всю ширину окна после панели навигации. Если изменить размер окна, то блок будет или расширяться, или сужаться. Четвертый блок (`footer`) предназначен для вывода информации об авторских правах, а также для различных логотипов и счетчиков. Для этого блока с помощью атрибута `clear` отменяется обтекание с обеих сторон.

У блочной верстки Web-страницы с помощью `float` есть один недостаток. Если уменьшить ширину окна Web-браузера, то блоки выстроятся по вертикали, один под другим. Чтобы избежать такого эффекта, мы указали внешний отступ слева (`185 px`). Благодаря этому, блоки всегда будут расположены по горизонтали, независимо от ширины окна Web-браузера.

Следует обратить внимание еще на один момент: для блочного элемента нельзя задать вертикальное выравнивание с помощью атрибута `vertical-align`. Чтобы добиться центрирования по вертикали, мы указали значение атрибута `line-height` равным высоте блока.

2.14.5. Позиционирование блока

Задать способ позиционирования блока позволяет атрибут `position`. Он может принимать одно из следующих значений:

- `static` — статическое позиционирование (значение по умолчанию). Положение элемента в окне Web-браузера определяется его положением в HTML-документе;
- `relative` — относительное позиционирование. Координаты отсчитываются относительно позиции, в которую Web-браузер поместил бы элемент, будь он статически позиционированным;

- `absolute` — абсолютное позиционирование. Координаты отсчитываются относительно левого верхнего угла ближайшего родительского элемента, который имеет позиционирование, отличное от статического;
- `fixed` — фиксированное позиционирование. Координаты отсчитываются относительно левого верхнего угла окна Web-браузера. При прокрутке содержимого окна блок не смещается;
- `sticky` — сочетание `relative` и `fixed`. Вначале используется относительное позиционирование. Как только элемент доходит до указанной позиции, применяется фиксированное позиционирование. В листинге 2.9 мы создали макет страницы. Чтобы увидеть эффект от применения атрибута `sticky`, добавим следующее правило стиля к уже существующим:

  ```
  nav   { position: sticky; top: 10px }
  ```

  Затем уменьшим высоту окна Web-браузера таким образом, чтобы появилась вертикальная полоса прокрутки. Теперь при перемещении бегунка полосы прокрутки вниз вначале панель навигации будет сдвигаться вверх, как обычный элемент страницы. Но как только до верха страницы останется 10 пикселов, панель навигации остановится, а все остальное содержимое страницы будет сдвигаться вверх и дальше. Таким образом панель навигации всегда будет видна независимо от положения полосы прокрутки.

Для указания привязки предназначены следующие атрибуты:

- `left` — расстояние от левой границы;
- `top` — расстояние от верхней границы;
- `right` — расстояние от правой границы;
- `bottom` — расстояние от нижней границы.

Эти атрибуты могут иметь отрицательные значения и значение `auto`. Статически позиционированные элементы не имеют атрибутов `left`, `top`, `right` и `bottom`.

Давайте рассмотрим все это на примере (листинг 2.10).

Листинг 2.10. Позиционирование блоков

```
<!DOCTYPE html>
<html lang="ru">
<head>
 <meta charset="utf-8">
 <title>Позиционирование блоков</title>
 <style type="text/css">
  body { font-family: Verdana, Tahoma, sans-serif; font-size: 14px }
  div { border: 1px solid black }
  .div1 { width: 10px; height: 2000px; left: 900px; top: 0;
          position: absolute }
  .div2 { height: 20px; position: static; background-color: silver }
  .div3 { height: 20px; position: relative; top: 30px;
          background-color: silver }
```

```
  .div4 { width: 150px; height: 150px; position: absolute; top: 30px;
          left: 400px; background-color: green }
  .div5 { width: 300px; height: 300px; position: absolute; top: 250px;
          left: 400px; }
  .div6 { width: 100px; height: 100px; position: absolute; top: 50px;
          left: 50px; background-color: #F5D8C1 }
  .div7 { width: 150px; height: 300px; position: fixed; top: 150px;
          left: 20px; background-color: #FF9600 }
  .div8 { width: 100%; height: 50px; left: 0; bottom: 0; margin: 0;
          position: fixed; background-color: #F4AB56 }
 </style>
</head>
<body>
 <div class="div1"></div>
 <div class="div2">Статическое позиционирование</div>
 <div class="div3">Относительное позиционирование</div>
 <div class="div4">Абсолютное позиционирование</div>
 <div class="div5">Абсолютное позиционирование внутри родительского блока
    <div class="div6">top: 50px; left: 50px;</div>
 </div>
 <div class="div7">Фиксированное позиционирование</div>
 <div class="div8">Фиксированное позиционирование относительно
                  нижней границы</div>
</body>
</html>
```

Как можно видеть, страница содержит восемь блоков:

- блок `div1` имеет абсолютное позиционирование и смещен на `900 px` относительно левой границы Web-страницы. Для блока также указана большая высота (`2000 px`). Это позволит увидеть эффект фиксированного позиционирования для блоков `div7` и `div8`, т. к. Web-браузер отобразит вертикальную полосу прокрутки;
- блок `div2` имеет статическое позиционирование, а блок `div3` — относительное. Блок `div3` сдвинут на `30 px` вниз относительно блока `div2`, а не от верхней границы Web-страницы;
- блоки `div4`, `div5` и `div6` имеют абсолютное позиционирование. Блок `div4` сдвинут на `400 px` относительно левой границы Web-страницы и на `30 px` — относительно верхней. Внутри блока `div5` расположен блок `div6`. Смещения этого блока отсчитываются относительно блока `div5`, а не границ Web-страницы;
- блоки `div7` и `div8` имеют фиксированное позиционирование. Блок `div7` демонстрирует возможность размещения панели навигации в определенном месте, а блок `div8` — прикрепление блока к нижней границе окна Web-браузера. Чтобы увидеть отличие этого вида позиционирования от других, переместите бегунок вертикальной полосы прокрутки вниз — эти блоки всегда остаются на своих местах и при прокрутке не перемещаются.

2.14.6. Последовательность отображения слоев

Порядок, в котором свободно позиционированные элементы будут перекрываться друг другом, устанавливает атрибут `z-index`: элемент с большим значением `z-index` перекрывает элементы с меньшим значением. Значение по умолчанию `auto`.

Рассмотрим порядок перекрытия на примере (листинг 2.11).

Листинг 2.11. Атрибут `z-index`

```
<!DOCTYPE html>
<html lang="ru">
<head>
 <meta charset="utf-8">
 <title>Атрибут z-index</title>
 <style type="text/css">
  body { font-family: Verdana, Tahoma, sans-serif; font-size: 11pt }
  #main { position: relative; top: 5px; height: 300px }
  #main div { width: 100px; height: 100px; position: absolute }
  #div1 { top: 70px;  left: 50px;  z-index: 1; background: silver }
  #div2 { top: 120px; left: 100px; z-index: 2; background: red }
  #div3 { top: 170px; left: 150px; z-index: 3; background: green }
 </style>
</head>
<body>
  <div id="main">
     <div id="div1">z-index = 1</div>
     <div id="div2">z-index = 2</div>
     <div id="div3">z-index = 3</div>
  </div>
</body>
</html>
```

2.15. Управление отображением элемента

Видимость элемента в окне Web-браузера задает атрибут `visibility`. Он может принимать следующие значения:

- `inherit` — если родитель видим, то видим и элемент (значение по умолчанию);
- `visible` — элемент отображается независимо от видимости родителя;
- `hidden` — элемент скрывается независимо от видимости родителя;
- `collapse` — полностью скрывает строку или столбец таблицы.

Невидимый элемент при использовании `hidden` все равно занимает место на Web-странице. Для того чтобы скрыть элемент и убрать его с Web-страницы, можно воспользоваться атрибутом `display` со значением `none`.

Атрибуты могут задавать лишь начальное поведение элемента. Отобразить же скрытый элемент можно только с помощью языка программирования JavaScript. Рассмотрим пример использования атрибутов `visibility` и `display`, а заодно познакомимся с языком программирования JavaScript (листинг 2.12).

Листинг 2.12. Скрытие и отображение элементов

```
<!DOCTYPE html>
<html lang="ru">
<head>
 <meta charset="utf-8">
 <title>Скрытие и отображение элементов</title>
 <script>
 function changeDisplay() {
   var elem1 = document.getElementById("elem1");
   if (elem1.style.display == "none") {
      elem1.style.display = "block";
   }
   else {
      elem1.style.display = "none";
   }
 }
 function changeVisibility() {
   var elem2 = document.getElementById("elem2");
   if (elem2.style.visibility == "hidden") {
      elem2.style.visibility = "visible";
   }
   else {
      elem2.style.visibility = "hidden";
   }
 }
 </script>
</head>
<body>
 <div><a href="#" onclick="changeDisplay(); return false">
 Щелкните на ссылке, чтобы отобразить или скрыть абзац</a></div>
 <p id="elem1" style="display: none; background-color: silver">
 Скрытый абзац</p>
 <p>Демонстрация применения атрибута display.</p>

 <div><a href="#" onclick="changeVisibility(); return false">
 Щелкните на ссылке, чтобы отобразить или скрыть абзац</a></div>
 <p id="elem2" style="visibility: hidden; background-color: silver">
 Скрытый абзац</p>
 <p>Демонстрация применения атрибута visibility.</p>
</body>
</html>
```

Итак, первая ссылка демонстрирует применение атрибута `display`. При щелчке на ссылке отображается скрытый абзац, и все содержимое страницы сдвигается вниз. При повторном щелчке абзац скрывается, и все содержимое страницы сдвигается вверх на место скрытого абзаца.

Вторая ссылка демонстрирует применение атрибута `visibility`. При щелчке на ссылке скрытый абзац отображается, а при повторном щелчке — скрывается. Но в отличие от того, что происходит при изменении атрибута `display`, в этом случае все остальное содержимое страницы остается на своих первоначальных местах.

2.16. Flex-контейнеры

При изучении атрибута `display` мы упомянули два значения, появившиеся в CSS 3, но оставили их без внимания:

- `flex` — блочный flex-контейнер;
- `inline-flex` — строчный flex-контейнер.

Элементы, для которых атрибут `display` имеет значение `flex`, являются *контейнерами*, внутри которых можно задавать направление выравнивания дочерних элементов, управлять растяжением или сжатием элементов, переносом элементов на новую строку и т. д.

2.16.1. Направление выравнивания элементов внутри контейнера

Направление выравнивания элементов внутри flex-контейнера задает атрибут `flex-direction`. Он может принимать следующие значения:

- `row` — выравнивание по горизонтали слева направо (значение по умолчанию);
- `row-reverse` — выравнивание по горизонтали справа налево;
- `column` — выравнивание по вертикали сверху вниз;
- `column-reverse` — выравнивание по вертикали снизу вверх.

Если тег имеет параметр `dir` со значением `"rtl"`, то направление выравнивания по горизонтали меняется на противоположное.

Пример выравнивания элементов по вертикали приведен в листинге 2.13. В дальнейших примерах мы будем менять содержимое тега `<style>`, не меняя разметки.

Листинг 2.13. Выравнивание элементов по вертикали внутри flex-контейнера

```
<!DOCTYPE html>
<html lang="ru">
<head>
 <meta charset="utf-8">
 <title>Flex-контейнеры</title>
 <style type="text/css">
```

```
   #main { position: relative; height: 350px; width: 500px;
           border: 3px black solid;
           display: flex; flex-direction: column }
   #main div { width: 100px; height: 100px; padding: 5px }
   #div1 { background: silver }
   #div2 { background: red }
   #div3 { background: green }
 </style>
</head>
<body>
  <div id="main">
     <div id="div1">div1</div>
     <div id="div2">div2</div>
     <div id="div3">div3</div>
  </div>
</body>
</html>
```

2.16.2. Перенос на новую строку

Управлять переносом элементов на новую строку позволяет атрибут `flex-wrap`. Он может принимать следующие значения:

- `nowrap` — элементы выстраиваются в одну линию. Если элементы не помещаются внутри контейнера, то производится уменьшение размеров элементов (значение по умолчанию);
- `wrap` — если элементы не помещаются внутри контейнера, то они переносятся на новую строку в направлении, заданном атрибутом `flex-direction`;
- `wrap-reverse` — если элементы не помещаются внутри контейнера, то они переносятся на новую строку в направлении, противоположном значению атрибута `flex-direction`.

В этом примере элементы будут отображаться на двух строках:

```
#main { position: relative; height: 350px; width: 250px;
        border: 3px black solid;
        display: flex; flex-direction: row; flex-wrap: wrap }
```

А в этом примере — на одной строке с уменьшением размеров элементов:

```
#main { position: relative; height: 350px; width: 250px;
        border: 3px black solid;
        display: flex; flex-direction: row; flex-wrap: nowrap }
```

2.16.3. Одновременное указание характеристик flex-контейнера

За один раз указать все характеристики flex-контейнера позволяет атрибут `flex-flow`:

```
flex-flow: [<direction>] [<wrap>]
```

Можно указать все значения:

```
#main { position: relative; height: 350px; width: 500px;
        border: 3px black solid;
        display: flex; flex-flow: row nowrap }
```

или только одно:

```
#main { position: relative; height: 350px; width: 500px;
        border: 3px black solid;
        display: flex; flex-flow: column }
```

2.16.4. Размеры элемента

Предпочтительную ширину элемента (при горизонтальном выравнивании) или его предпочтительную высоту (при вертикальном выравнивании) задает атрибут `flex-basis`. Если свободного места достаточно, то элемент будет иметь указанный размер, а если нет, то его размер будет определяться значением атрибута `flex-shrink` (см. далее). Если атрибут `flex-basis` не указан, то он по умолчанию будет иметь значение `auto`, означающее, что размер определяется содержимым элемента:

```
#main { position: relative; height: 350px; width: 500px;
        border: 3px black solid; display: flex; flex-direction: row }
#main div { padding: 5px }
#div1 { background: silver; flex-basis: auto }
#div2 { background: red; flex-basis: 100px }
#div3 { background: green; flex-basis: 200px }
```

2.16.5. Растяжение элементов

Фактор растяжения при наличии свободного места в контейнере задает атрибут `flex-grow`. По умолчанию атрибут имеет значение `0`, означающее, что размер элемента будет соответствовать предпочтительным размерам, например, заданным с помощью атрибута `flex-basis`. Значение атрибута `flex-grow` можно сравнить с жесткостью пружины, вставленной внутрь элемента: чем больше значение, тем больше места будет занимать элемент. Если для двух элементов задано одинаковое значение, то их размеры будут равны:

```
#main { position: relative; height: 350px; width: 500px;
        border: 3px black solid;
        display: flex; flex-direction: column }
#main div { padding: 5px }
#div1 { background: silver; flex-basis: 50px }
#div2 { background: red; flex-basis: 50px }
#div3 { background: green; flex-basis: 50px; flex-grow: 1 }
```

В этом примере элементы выстраиваются по вертикали сверху вниз. Для всех элементов задана предпочтительная высота `50 px`. Для элемента `div3` дополнительно указан фактор растяжения `1`. Все остальные элементы имеют фактор растяжения,

равный `0`. Свободной высоты контейнера достаточно, поэтому первые два элемента будут иметь высоту `50 px`, а элемент `div3` будет растянут на всю оставшуюся высоту, т. к. имеет фактор растяжения (пружину внутри себя).

2.16.6. Сжатие элементов

Сжатием размеров элемента при отсутствии свободного пространства управляет атрибут `flex-shrink`. По умолчанию атрибут имеет значение `1`, означающее, что при недостатке места размер элемента будет уменьшаться. Значение больше `1` задает фактор сжатия: чем больше значение, тем сильнее сжимается элемент, освобождая пространство для других элементов. Если нужно, чтобы размеры элемента были равны предпочтительным, то следует указать значение `0`:

```
#main { position: relative; height: 350px; width: 250px;
        border: 3px black solid; display: flex; flex-direction: row }
#main div { padding: 5px }
#div1 { background: silver; flex-basis: 100px; flex-shrink: 0 }
#div2 { background: red; flex-basis: 100px }
#div3 { background: green; flex-basis: 100px; flex-shrink: 3 }
```

В этом примере элемент `div1` будет иметь предпочтительную ширину `100 px`, т. к. атрибут `flex-shrink` имеет значение `0`. Оставшееся пространство будет разделено между элементами `div2` и `div3`. Причем ширина элемента `div2` будет больше ширины элемента `div3`, т. к. фактор сжатия у него меньше.

2.16.7. Одновременное указание характеристик элементов

За один раз указать все характеристики элементов внутри flex-контейнера позволяет атрибут `flex`. Делается это по следующей схеме:

```
flex: [<grow>[ <shrink>] ]<basis>
```

Если указано одно значение, то оно задает значение атрибуту `flex-basis`:

```
#div2 { background: red; flex: 100px }
```

Если указаны два значения, то первое задает значение атрибуту `flex-grow`, а второе — атрибуту `flex-basis`:

```
#div1 { background: silver; flex: 2 100px }
```

Вот пример указания всех значений:

```
#div3 { background: green; flex: 0 3 100px }
```

Можно также указать значение `none`, соответствующее значению `0 0 auto`:

```
#div3 { background: green; flex: none }
```

2.16.8. Выравнивание элементов внутри контейнера

Выравнивание элементов внутри flex-контейнера (по ширине для горизонтального контейнера или по высоте для вертикального) задает атрибут `justify-content`. Он может принимать следующие значения:

- `flex-start` — выравнивание по началу контейнера (значение по умолчанию);
- `flex-end` — выравнивание по концу контейнера;
- `center` — выравнивание по центру контейнера;
- `space-between` — первый элемент прижат к началу контейнера, последний — к концу, а остальные равномерно распределяются внутри свободного пространства;
- `space-around` — равномерное выравнивание элементов, но перед первым и после последнего расстояние в два раза меньше, чем между элементами;
- `space-evenly` — равномерное выравнивание элементов с одинаковыми пустыми пространствами.

Пример:

```
#main { position: relative; height: 350px; width: 500px;
        border: 3px black solid; display: flex;
        flex-flow: row; justify-content: space-between }
#main div { width: 100px; height: 100px; padding: 5px }
#div1 { background: silver }
#div2 { background: red }
#div3 { background: green }
```

Выравнивание элементов внутри flex-контейнера (по высоте для горизонтального контейнера или по ширине для вертикального) задает атрибут `align-content` — при условии, что атрибут `flex-wrap` имеет значение, отличное от `nowrap`. Атрибут может принимать следующие значения: `stretch` (значение по умолчанию), `flex-start`, `flex-end`, `center`, `space-between`, `space-around` и `space-evenly`. Смысл значений точно такой же, как и у одноименных значений атрибута `justify-content`, — различие только в направлении выравнивания.

Вот пример горизонтального выравнивания по центру и равномерного выравнивания по вертикали для горизонтального контейнера:

```
#main { position: relative; height: 350px; width: 250px;
        border: 3px black solid; display: flex;
        flex-flow: row wrap; align-content: space-around;
        justify-content: center }
#main div { width: 100px; height: 100px; padding: 5px }
#div1 { background: silver }
#div2 { background: red }
#div3 { background: green }
```

Если для элемента горизонтального контейнера не задана высота, то при указании значения `stretch` элемент будет растягиваться по высоте:

```
#main { position: relative; height: 350px; width: 250px;
        border: 3px black solid; display: flex;
        flex-flow: row wrap; align-content: stretch }
#main div { width: 100px; padding: 5px }
#div1 { background: silver }
#div2 { background: red }
#div3 { background: green }
```

Выравнивание элементов внутри строки (по высоте для горизонтального контейнера или по ширине для вертикального) задает атрибут `align-items`. Он может принимать следующие значения: `stretch` (значение по умолчанию), `flex-start`, `flex-end`, `center` и `baseline` (по базовой линии). Смысл значений точно такой же, как и у одноименных значений атрибута `justify-content`, — различие только в направлении выравнивания. Обратите внимание: атрибут `align-items` задает выравнивание внутри строки, а атрибут `align-content` — внутри flex-контейнера:

```
#main { position: relative; height: 350px; width: 250px;
        border: 3px black solid; display: flex;
        flex-flow: row wrap; align-items: center;
        justify-content: center; align-content: space-evenly }
#main div { width: 100px; padding: 5px }
#div1 { background: silver; height: 50px }
#div2 { background: red; height: 100px }
#div3 { background: green; height: 80px }
```

Если для элемента горизонтального контейнера не задана высота, то при указании значения `stretch` элемент будет растягиваться по высоте, что позволяет делать столбцы одинаковой высоты независимо от содержимого:

```
#main { position: relative; height: 350px; width: 500px;
        border: 3px black solid; display: flex;
        flex-flow: row nowrap; align-items: stretch;
        justify-content: center }
#main div { width: 33%; padding: 5px; margin: 10px }
#div1 { background: silver }
#div2 { background: red }
#div3 { background: green }
```

Выравнивание отдельного элемента внутри строки (по высоте для горизонтального контейнера или по ширине для вертикального) задает атрибут `align-self`. Он может принимать следующие значения (переопределяет значение атрибута `align-items`): `auto` (значение по умолчанию), `stretch`, `flex-start`, `flex-end`, `center` и `baseline` (по базовой линии):

```
#main { position: relative; height: 350px; width: 250px;
        border: 3px black solid; display: flex;
        flex-flow: row wrap; align-items: center;
        justify-content: center; align-content: space-evenly }
```

```
#main div { width: 100px; padding: 5px }
#div1 { background: silver; height: 50px; align-self: flex-end }
#div2 { background: red; height: 100px }
#div3 { background: green; height: 80px }
```

2.16.9. Порядок следования элементов внутри контейнера

Порядок следования элементов внутри контейнера позволяет задать атрибут `order`. В качестве его значения можно указать положительное или отрицательное число: чем больше число, тем дальше будет расположен элемент. По умолчанию и при равных значениях атрибута элементы отображаются в порядке добавления в контейнер.

Переместим первый добавленный элемент в самый конец контейнера:

```
#main { position: relative; height: 350px; width: 500px;
        border: 3px black solid; display: flex;
        flex-direction: row }
#main div { height: 100px; flex-basis: 100px; padding: 5px }
#div1 { background: silver; order: 3 }
#div2 { background: red; order: 1 }
#div3 { background: green; order: 2 }
```

2.17. CSS Grid

Атрибут `display` может также принимать следующие значения, появившиеся в CSS 3 и поддерживаемые самыми новыми версиями Web-браузеров (поэтому не торопитесь пока использовать CSS Grid, т. к. получите проблемы со старыми версиями):

- `grid` — блочный grid-контейнер;
- `inline-grid` — строчный grid-контейнер;
- `subgrid` — вложенный grid-контейнер.

Элементы, для которых атрибут `display` имеет значение `grid`, являются контейнерами, представляющими сетку с ячейками, внутри которых можно размещать элементы.

2.17.1. Описание строк и столбцов

Характеристики столбцов сетки задает атрибут `grid-template-columns`:

```
grid-template-columns: <Размер 1> ... <Размер N>
```

Размеры можно задать в абсолютных величинах, процентах, с помощью ключевого слова `auto`, а также в единицах свободного пространства в сетке — `fr`. Если указано значение `none`, то текущие характеристики сбрасываются.

Вот пример описания двух столбцов одинаковой ширины:

```
grid-template-columns: 1fr 1fr;
```

А здесь второй столбец будет иметь ширину, в два раза большую ширины первого столбца:

```
grid-template-columns: 1fr 2fr;
```

В следующем примере первый столбец будет иметь ширину `100 px`, а второй столбец займет всю оставшуюся ширину:

```
grid-template-columns: 100px auto;
```

Если размеры для нескольких рядом расположенных столбцов одинаковы, то можно воспользоваться функцией `repeat()`:

```
repeat(<Число повторов> | auto-fill | auto-fit, <Характеристики>)
```

Вот пример указания характеристик двух столбцов:

```
grid-template-columns: repeat(2, 50%);
```

Указать диапазон допустимых значений позволяет функция `minmax()`:

```
minmax(<Минимум>, <Максимум>)
```

Если ширина контейнера позволяет, то второй столбец будет иметь ширину `200 px`, а при уменьшении размеров контейнера предпочтительная ширина столбца будет в диапазоне между `100 px` и `200 px`:

```
grid-template-columns: 1fr minmax(100px, 200px);
```

В качестве размера можно указать ключевые слова `min-content` и `max-content` — в этом случае минимальный и максимальный размеры определяются содержимым элемента:

```
grid-template-columns: 1fr minmax(min-content, max-content);
```

Если в качестве минимального значения указано ключевое слово `auto`, то размер столбца определяется минимальной шириной элемента (значением атрибута `min-width`). Если ключевое слово `auto` указано для максимального значения, то оно эквивалентно `max-content`:

```
grid-template-columns: 1fr minmax(auto, auto);
```

Задать размер можно с помощью функции `fit-content()`:

```
grid-template-columns: 1fr fit-content(300px);
```

Для динамического управления количеством столбцов в зависимости от ширины контейнера можно воспользоваться значениями `auto-fill` и `auto-fit` в сочетании с функциями `repeat()` и `minmax()`:

```
grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
```

При изменении размеров контейнера количество столбцов будет меняться.

Внутри квадратных скобок можно дать названия линиям, разделяющим ячейки:

```
grid-template-columns: [vline1] 1fr [vline2] 1fr [vline3];
```

Если линии не имеют имен, то можно обращаться к ним по индексу (нумерация начинается с единицы).

Характеристики строк сетки задает атрибут `grid-template-rows`:

```
grid-template-rows: <Размер 1> ... <Размер N>
```

Характеристики строк задаются точно так же, как и характеристики столбцов:

```
grid-template-rows: 1fr 1fr;
grid-template-rows: 30% 70%;
grid-template-rows: repeat(2, 50%);
grid-template-rows: [hline1] 1fr [hline2] 1fr [hline3];
```

Можно также просто указать значение `auto`, чтобы Web-браузер автоматически подбирал характеристики строк:

```
grid-template-rows: auto;
```

Задать характеристики строк и столбцов одновременно позволяет атрибут `grid-template`. Его формат:

```
grid-template: <grid-template-rows> / <grid-template-columns>
```

А вот пример его использования:

```
grid-template: repeat(2, 1fr) / repeat(5, 1fr);
```

2.17.2. Автоматическое размещение элементов внутри сетки

На самом деле, нам даже не нужно описывать характеристики строк и столбцов. Как только мы задали атрибуту `display` для контейнера значение `grid`, все дочерние элементы будут автоматически распределены по ячейкам сетки:

```
<div id="main" style="display: grid; height: 350px; width: 500px">
   <div id="div1">div1</div>
   <div id="div2">div2</div>
   <div id="div3">div3</div>
   <div id="div4">div4</div>
</div>
```

В этом примере сетка состоит из одного столбца и четырех строк, и ячейки имеют одинаковые размеры.

Если дополнительно указать характеристики столбцов, то элементы разместятся по ячейкам слева направо и сверху вниз. Так, в следующем примере сетка состоит из трех столбцов и двух строк:

```
<div id="main" style="display: grid; height: 350px; width: 500px;
                      grid-template-columns: repeat(3, 1fr)">
   <div id="div1">div1</div>
   <div id="div2">div2</div>
```

```
   <div id="div3">div3</div>
   <div id="div4">div4</div>
</div>
```

Управлять автоматическим размещением элементов в ячейках сетки позволяет атрибут `grid-auto-flow`. Он может принимать следующие значения:

- `row` — по строкам слева направо и сверху вниз (значение по умолчанию);
- `column` — по столбцам сверху вниз и слева направо;
- `dense` — при размещении будут заполняться дыры в сетке, поэтому элементы могут идти не по порядку. Можно указать комбинации значений: `row dense` и `column dense`.

Вот пример размещения по столбцам сверху вниз и слева направо:

```
<div id="main" style="display: grid; height: 350px; width: 500px;
                      grid-template-rows: repeat(3, 1fr);
                      grid-auto-flow: column">
   <div id="div1">div1</div>
   <div id="div2">div2</div>
   <div id="div3">div3</div>
   <div id="div4">div4</div>
</div>
```

2.17.3. Добавление элементов в ячейки сетки

Добавить элемент в определенную ячейку позволяют следующие атрибуты:

- `grid-column-start` — задает начальную вертикальную линию (нумерация линий начинается с единицы);
- `grid-column-end` — задает конечную вертикальную линию:

  ```
  grid-column-start: 1; grid-column-end: 2;
  ```

- `grid-column` — позволяет указать сразу два значения одновременно:

  ```
  grid-column: <grid-column-start> [/ <grid-column-end>]
  ```

  Пример:

  ```
  grid-column: 1 / 2;
  ```

  Если элемент занимает только одну ячейку, то можно указать лишь начальную линию, т. к. по умолчанию атрибуты имеют значение `auto`:

  ```
  grid-column: 1;
  ```

- `grid-row-start` — задает начальную горизонтальную линию;
- `grid-row-end` — задает конечную горизонтальную линию:

  ```
  grid-row-start: 1; grid-row-end: 2;
  ```

- `grid-row` — позволяет указать сразу два значения одновременно:

  ```
  grid-row: <grid-row-start> [/ <grid-row-end>]
  ```

Примеры:

```
grid-row: 1 / 2;
grid-row: 1;
```

- `grid-area` — позволяет указать сразу все значения одновременно:

```
grid-area: <grid-row-start> / <grid-column-start> /
           <grid-row-end> / <grid-column-end>
```

Вот пример добавления элемента в первую ячейку:

```
grid-area: 1 / 1 / 2 / 2;
```

Пример распределения элементов по ячейкам сетки приведен в листинге 2.14. В дальнейших примерах мы будем менять содержимое тега `<style>`, не меняя разметки.

Листинг 2.14. Добавление элементов в ячейки сетки

```
<!DOCTYPE html>
<html lang="ru">
<head>
 <meta charset="utf-8">
 <title>CSS Grid</title>
 <style type="text/css">
   #main { height: 350px; width: 500px; display: grid;
           grid-template-columns: repeat(2, 50%);
           grid-template-rows: repeat(2, 50%) }
   #main div { padding: 5px }
   #div1 { background: silver; grid-row: 1; grid-column: 1 }
   #div2 { background: red; grid-row: 1; grid-column: 2 }
   #div3 { background: green; grid-row: 2; grid-column: 1 }
   #div4 { background: navy; grid-row: 2; grid-column: 2 }
 </style>
</head>
<body>
  <div id="main">
     <div id="div1">div1</div>
     <div id="div2">div2</div>
     <div id="div3">div3</div>
     <div id="div4">div4</div>
  </div>
</body>
</html>
```

Как вы уже знаете, при описании характеристик столбцов и строк можно дать линиям имена. Эти имена допускается указывать вместо индексов:

```
#main { height: 350px; width: 500px; display: grid;
        grid-template-columns: [vline1] 1fr [vline2] 1fr [vline3];
        grid-template-rows: [hline1] 1fr [hline2] 1fr [hline3] }
```

```
#main div { padding: 5px }
#div1 { background: silver; grid-row: hline1; grid-column: vline1 }
#div2 { background: red; grid-row: hline1; grid-column: vline2 }
#div3 { background: green; grid-row: hline2; grid-column: vline1 }
#div4 { background: navy; grid-row: hline2; grid-column: vline2 }
```

2.17.4. Объединение ячеек

Один элемент может занимать сразу несколько ячеек сетки. Для объединения ячеек в атрибутах `grid-column-end` и `grid-row-end` указывается конечная линия или ключевое слово `span`, после которого задается число объединяемых ячеек или название конечной линии:

```
grid-column-start: 4; grid-column-end: span 2;
grid-row-start: 1; grid-row-end: 3;
```

В атрибутах `grid-column` и `grid-row` значение задается после символа `/`:

```
#main { height: 350px; width: 600px; display: grid;
        grid-template-columns: repeat(5, 1fr);
        grid-template-rows: auto }
#main div { padding: 5px }
#div1 { background: silver; grid-row: 1 / span 2; grid-column: 1 }
#div2 { background: red; grid-row: 1; grid-column: 2 / 4 }
#div3 { background: green; grid-row: 2; grid-column: 2 / 4 }
#div4 { background: navy; grid-row: 1 / 3; grid-column: 4 / span 2 }
```

В этом примере элемент `div1` будет занимать две ячейки по вертикали, элементы `div2` и `div3` — по две ячейки по горизонтали, а элемент `div4` — две ячейки по горизонтали и две ячейки по вертикали.

2.17.5. Размеры неявных ячеек

Задавая позицию вставки элемента в ячейку, допускается указывать индексы несуществующих линий, — в итоге элемент будет расположен в ячейке, характеристики которой мы не описывали. Размеры промежуточных неявных ячеек по умолчанию будут равны нулю. С помощью атрибутов `grid-auto-columns` и `grid-auto-rows` можно задать размеры таких неявных ячеек:

```
#main { height: 500px; width: 800px; display: grid;
        grid-template-columns: repeat(2, 1fr);
        grid-template-rows: repeat(2, 1fr);
        grid-auto-columns: 50px; grid-auto-rows: 50px }
#main div { padding: 5px }
#div1 { background: silver; grid-row: 1; grid-column: 1 }
#div2 { background: red; grid-row: 1; grid-column: 2 }
#div3 { background: green; grid-row: 2; grid-column: 1 }
#div4 { background: navy; grid-row: 4; grid-column: 5 }
```

Чтобы увидеть границы ячеек сетки, открываем панель **Инструменты разработчика** Web-браузера Firefox, переходим на вкладку **Инспектор** и выделяем строку с кодом контейнера. Затем справа на вкладке **Правила** находим атрибут `display` и щелкаем на значке с изображением сетки левой кнопкой мыши — границы ячеек в окне Web-браузера станут видны. Повторный щелчок на значке скрывает эти границы. Можно также перейти на вкладку **Разметка | Сетка** и установить все флажки (рис. 2.4) — вы увидите линии, номера линий, названия элементов и др.

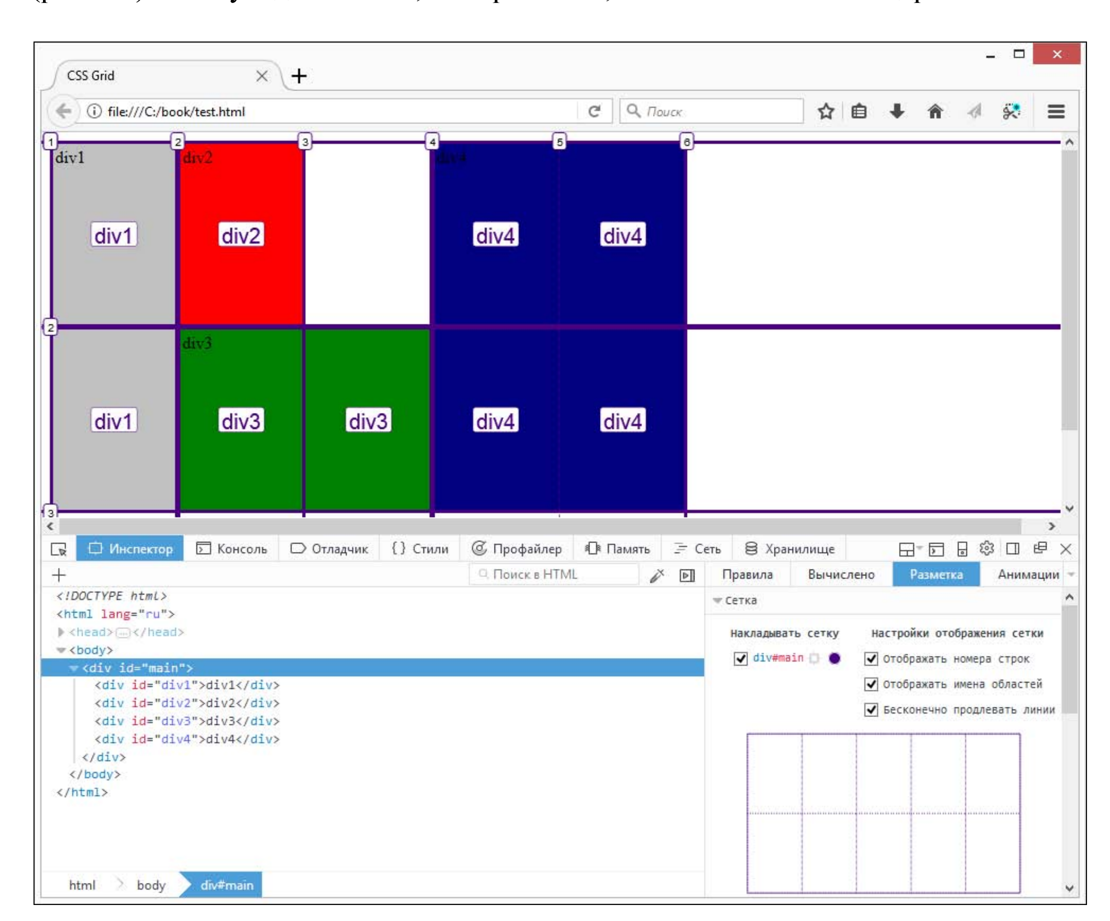


Рис. 2.4. Отображение характеристик сетки в Web-браузере Firefox

2.17.6. Расстояние между ячейками

Если размеры элементов не заданы, то они будут заполнять всю ячейку. При этом между ячейками не останется никакого свободного пространства. Указать размеры отступов (точнее — ширину внутренних линий) позволяют атрибуты `grid-row-gap` и `grid-column-gap`. По умолчанию они имеют значение `0`:

```
#main { height: 350px; width: 500px; display: grid;
        grid-template-columns: repeat(2, 1fr);
```

```
        grid-template-rows: repeat(2, 1fr);
        grid-column-gap: 10px; grid-row-gap: 10px }
#main div { padding: 5px }
#div1 { background: silver }
#div2 { background: red }
#div3 { background: green }
#div4 { background: navy }
```

Указать сразу все размеры отступов позволяет атрибут `grid-gap`:

```
grid-gap: <Размер>
grid-gap: <grid-row-gap> <grid-column-gap>
```

Если указано только одно значение, то оно задает размер отступов по горизонтали и вертикали:

```
grid-gap: 20px;
```

Если указано два значения, то первое задает отступ по горизонтали, а второе — отступ по вертикали:

```
grid-gap: 10px 20px;
```

2.17.7. Имена элементов

Каждому элементу внутри контейнера с помощью атрибута `grid-area` можно дать имя. В дальнейшем это имя указывается при размещении элементов в качестве значения для ячейки в атрибуте `grid-template-areas`. Этот атрибут позволяет наглядно разместить элементы внутри ячеек сетки. Для указания пустой ячейки нужно вставить точку, а для неопределенных областей используется значение `none`.

Вот пример размещения элементов с объединением ячеек:

```
#main { height: 350px; width: 600px; display: grid;
        grid-template-columns: repeat(5, 1fr);
        grid-template-rows: repeat(2, 1fr);
        grid-template-areas:
           "div1 div2   .  div4 div4"
           "div1 div3 div3 div4 div4"; }
#main div { padding: 5px }
#div1 { background: silver; grid-area: div1 }
#div2 { background: red; grid-area: div2 }
#div3 { background: green; grid-area: div3 }
#div4 { background: navy; grid-area: div4 }
```

Удобный формат для одновременного указания характеристик строк и столбцов, а также наглядного размещения именованных элементов по ячейкам, имеет атрибут `grid-template`:

```
#main { height: 350px; width: 600px; display: grid;
        grid-template: "div1 div2   .  div4 div4" 1fr
                       "div1 div3 div3 div4 div4" 1fr
                     /  1fr  1fr  1fr  1fr  1fr; }
```

```
#main div { padding: 5px }
#div1 { background: silver; grid-area: div1 }
#div2 { background: red; grid-area: div2 }
#div3 { background: green; grid-area: div3 }
#div4 { background: navy; grid-area: div4 }
```

2.17.8. Одновременное указание характеристик контейнера

За один раз указать все характеристики контейнера позволяет атрибут `grid`:

```
grid: <grid-template>
grid: <grid-template-rows> / auto-flow[ dense] [<grid-auto-columns>]
grid: auto-flow[ dense] [<grid-auto-rows>] / <grid-template-columns>
grid: none
```

Вот примеры его использования:

```
grid: repeat(2, 1fr) / repeat(5, 1fr);
grid: "div1 div2   .  div4 div4" 1fr
      "div1 div3 div3 div4 div4" 1fr
    /  1fr  1fr  1fr  1fr  1fr;
grid: 1fr 1fr / auto-flow 1fr;
grid: 1fr 1fr / auto-flow dense;
grid: auto-flow 1fr / 1fr 1fr;
grid: auto-flow / 1fr 1fr;
```

2.17.9. Выравнивание сетки внутри контейнера

Выравнивание сетки внутри контейнера по горизонтали задает атрибут `justify-content`. Он может принимать следующие значения:

- `start` — выравнивание по левой стороне контейнера (значение по умолчанию);
- `end` — выравнивание по правой стороне контейнера;
- `center` — выравнивание по центру контейнера;
- `space-between` — первый столбец прижат к левой стороне контейнера, последний — к правой стороне, а остальные равномерно распределяются внутри свободного пространства;
- `space-around` — равномерное выравнивание столбцов, но перед первым и после последнего расстояние в два раза меньше, чем между другими столбцами;
- `space-evenly` — равномерное выравнивание столбцов с одинаковыми пустыми пространствами.

Пример:

```
#main { height: 350px; width: 500px; display: grid;
        grid-template-columns: repeat(3, 100px);
        grid-template-rows: repeat(2, 100px);
```

```
        border: 2px blue solid;
        justify-content: space-between }
#main div { height: 50px; width: 50px; padding: 5px }
#div1 { background: silver }
#div2 { background: red }
#div3 { background: green }
#div4 { background: navy }
```

Выравнивание сетки внутри контейнера по вертикали задает атрибут `align-content`. Он может принимать следующие значения: `start`, `end`, `center`, `space-between`, `space-around` и `space-evenly`. Смысл значений точно такой же, как и у одноименных значений атрибута `justify-content`, — различие только в направлении выравнивания.

Вот пример горизонтального выравнивания по центру и равномерного выравнивания по вертикали:

```
#main { height: 350px; width: 500px; display: grid;
        grid-template-columns: repeat(2, 100px);
        grid-template-rows: repeat(2, 100px);
        border: 2px blue solid;
        justify-content: center; align-content: space-around }
#main div { height: 50px; width: 50px; padding: 5px }
#div1 { background: silver }
#div2 { background: red }
#div3 { background: green }
#div4 { background: navy }
```

2.17.10. Выравнивание элемента внутри ячейки

Выравнивание элемента внутри ячейки сетки по горизонтали задает атрибут `justify-self`. Он может принимать следующие значения:

- `start` — выравнивание по левой стороне ячейки;
- `end` — выравнивание по правой стороне ячейки;
- `center` — выравнивание по центру ячейки;
- `stretch` — если для элемента не задана ширина, то растягивает элемент на всю ширину ячейки.

Пример:

```
#main { height: 350px; width: 500px; display: grid;
        grid-template-columns: repeat(2, 1fr);
        grid-template-rows: repeat(2, 1fr);
        border: 2px blue solid }
#main div { height: 50px; padding: 5px }
#div1 { background: silver; width: 50px; justify-self: start }
#div2 { background: red; width: 50px; justify-self: end }
#div3 { background: green; width: 50px; justify-self: center }
#div4 { background: navy; justify-self: stretch }
```

Для выравнивания всех элементов внутри ячеек по горизонтали предназначен атрибут `justify-items`. Обратите внимание: атрибут указывается для контейнера, а не для элемента.

Вот пример выравнивания по центру:

```
#main { height: 350px; width: 500px; display: grid;
        grid-template-columns: repeat(2, 1fr);
        grid-template-rows: repeat(2, 1fr);
        border: 2px blue solid; justify-items: center }
#main div { height: 50px; width: 50px; padding: 5px }
#div1 { background: silver }
#div2 { background: red }
#div3 { background: green }
#div4 { background: navy }
```

Выравнивание элемента внутри ячейки по вертикали задает атрибут `align-self`. Он может принимать следующие значения: `stretch`, `start`, `end`, `center`:

```
#main { height: 350px; width: 500px; display: grid;
        grid-template-columns: repeat(2, 1fr);
        grid-template-rows: repeat(2, 1fr);
        border: 2px blue solid }
#main div { width: 50px; padding: 5px }
#div1 { background: silver; height: 50px; align-self: start }
#div2 { background: red; height: 50px; align-self: center }
#div3 { background: green; height: 50px; align-self: end }
#div4 { background: navy; align-self: stretch }
```

Для выравнивания всех элементов внутри ячеек по вертикали предназначен атрибут `align-items`. Обратите внимание: атрибут указывается для контейнера, а не для элемента.

Вот пример выравнивания по центру:

```
#main { height: 350px; width: 500px; display: grid;
        grid-template-columns: repeat(2, 1fr);
        grid-template-rows: repeat(2, 1fr);
        border: 2px blue solid; align-items: center }
#main div { height: 50px; width: 50px; padding: 5px }
#div1 { background: silver }
#div2 { background: red }
#div3 { background: green }
#div4 { background: navy }
```

2.18. Многоколоночный текст

В последних версиях Web-браузеров появилась возможность вывода текста в несколько колонок. Так, например, Firefox версии 56 поддерживает многоколоночный текст без необходимости добавления к именам атрибутов «вендорных» префиксов.

Однако для поддержки старых версий Web-браузеров «вендорные» префиксы все же указывать нужно.

2.18.1. Количество колонок

Количество колонок задает атрибут `column-count`. В качестве его значения можно указать конкретное число или ключевое слово `auto` (значение по умолчанию), означающее, что количество колонок определяется автоматически на основе значений атрибутов `column-width` и `column-gap`:

```
p { column-count: 3 }
```

Одновременно указать количество колонок и/или их ширину можно с помощью атрибута `columns`:

```
columns: <column-width> <column-count>
```

Порядок следования значений может быть произвольным.

Приведем несколько примеров:

- создание трех колонок с предпочтительной шириной `200 px`:

  ```
  p { columns: 3 200px }
  ```

- указание только предпочтительной ширины колонок (количество колонок определяется динамически в зависимости от ширины области):

  ```
  p { columns: 200px }
  ```

- указание фиксированного количества колонок:

  ```
  p { columns: 3 }
  ```

2.18.2. Размеры колонок

Предпочтительную ширину колонок задает атрибут `column-width`. В качестве его значения можно указать конкретное число или ключевое слово `auto` (значение по умолчанию), означающее, что ширина колонок определяется автоматически на основе значений атрибутов `column-count` и `column-gap`:

```
p { column-width: 200px }
```

В этом примере количество колонок будет определяться шириной элемента.

Выравнивание текста внутри колонок по высоте задает атрибут `column-fill`. Он может принимать следующие значения:

- `balance` — колонки будут иметь одинаковую высоту (значение по умолчанию):

  ```
  p { column-width: 200px; column-fill: balance }
  ```

- `auto` — текст последовательно заполняет колонки по порядку. Если высота позволяет, то весь текст будет выведен в одну колонку:

  ```
  p { height: 200px; column-width: 200px; column-fill: auto }
  ```

2.18.3. Расстояние между колонками

Расстояние между колонками задает атрибут `column-gap`. В качестве его значения можно указать конкретное число или ключевое слово `normal` (значение по умолчанию), означающее, что расстояние между колонками определяет Web-браузер.

Приведем пару примеров:

- динамическое определение количества колонок:

```
p { column-width: 200px; column-gap: 20px }
```

- фиксированное количество колонок:

```
p { column-count: 3; column-gap: 20px }
```

2.18.4. Линии между колонками

Задать характеристики линий между колонками позволяют следующие атрибуты:

- `column-rule-style` — задает стиль линии. Может принимать те же самые значения, что и атрибут `border-style` (см. *разд. 2.7.1*). Значение по умолчанию: `none`;
- `column-rule-color` — задает цвет линии;
- `column-rule-width` — задает ширину линии:

```
p { column-width: 200px; column-gap: 30px;
    column-rule-style: solid; column-rule-color: red;
    column-rule-width: 2px }
```

- `column-rule` — позволяет указать все характеристики за один раз:

```
column-rule: <width> <style> <color>
```

Пример:

```
p { column-width: 200px; column-gap: 30px;
    column-rule: 5px dotted green }
```

2.19. Фильтры и эффекты

К элементам страницы мы можем применить различные фильтры и эффекты. Следует учитывать, что Internet Explorer содержит свои собственные фильтры и не поддерживает описанные в этом разделе возможности.

Применить фильтры к элементу позволяет атрибут `filter`:

```
filter: <Функция 1>[ ... <Функция N>]
```

Можно указать только одну функцию:

```
#img1 { filter: opacity(0.5) }
```

или сразу несколько функций через пробел:

```
#img1 { filter: opacity(0.5) sepia(1) }
...
<img alt="" src="photo.jpg" id="img1">
```

Значение `none` удаляет все фильтры:

```
#img1 { filter: none }
```

2.19.1. Изменение прозрачности

Изменить прозрачность элемента позволяет функция `opacity(<Значение>)`. В качестве параметра можно указать вещественное число от `0` (элемент полностью прозрачный) до `1` (элемент полностью непрозрачный) или проценты от `0%` (элемент полностью прозрачный) до `100%` (элемент полностью непрозрачный).

Сделаем изображение полупрозрачным, а при наведении курсора мыши плавно сделаем его полностью непрозрачным:

```
#img1       { filter: opacity(0.5); transition: 1s }
#img1:hover { filter: opacity(1) }
```

В этом примере плавность перехода между двумя состояниями элемента обеспечивает атрибут `transition`.

Вместо функции `opacity()` можно воспользоваться атрибутом `opacity`, который поддерживают все Web-браузеры. В качестве его значения указывается вещественное число от `0` (элемент полностью прозрачный) до `1` (элемент полностью непрозрачный):

```
#img1       { opacity: .5; transition: 1s }
#img1:hover { opacity: 1 }
```

2.19.2. Размытие

Функция `blur(<Значение>)` позволяет выполнить размытие по Гауссу. В качестве его параметра указывается радиус в абсолютных величинах. Чем больше значение, тем сильнее размытие:

```
#img1       { filter: blur(3px); transition: 1s }
#img1:hover { filter: blur(0) }
```

2.19.3. Изменение яркости, насыщенности и контраста

Изменить яркость, насыщенность и контраст позволяют следующие функции:

- `brightness(<Значение>)` — изменяет яркость изображения. Значение `1` или `100%` соответствует исходному изображению. Меньшее значение делает изображение темнее, а большее — светлее. Значение `0` делает изображение черным:

  ```
  #img1       { filter: brightness(50%); transition: 1s }
  #img1:hover { filter: brightness(100%) }
  ```

- saturate(<Значение>) — изменяет насыщенность изображения. Значение 1 или 100% соответствует исходному изображению. Меньшее значение делает изображение менее насыщенным, а большее — более насыщенным. Значение 0 делает изображение черно-белым:

```
#img1       { filter: saturate(0); transition: 1s }
#img1:hover { filter: saturate(1) }
```

- contrast(<Значение>) — изменяет контраст изображения. Значение 1 или 100% соответствует исходному изображению. Меньшее значение делает изображение менее контрастным, а большее — более контрастным. Значение 0 делает изображение серым:

```
#img1       { filter: contrast(.7); transition: 1s }
#img1:hover { filter: contrast(1) }
```

2.19.4. Изменение цвета

Изменить цвет изображения позволяют следующие функции:

- grayscale(<Значение>) — делает изображение черно-белым. Значение 0 соответствует исходному изображению. Значение 1 или 100% делает изображение черно-белым:

```
#img1       { filter: grayscale(100%); transition: 1s }
#img1:hover { filter: grayscale(0) }
```

- hue-rotate(<Значение>) — изменяет цветовой тон изображения. В качестве параметра указывается значение угла на цветом круге (от 0 до 360 deg):

```
#img1       { filter: hue-rotate(180deg); transition: 1s }
#img1:hover { filter: hue-rotate(0deg) }
```

- sepia(<Значение>) — создает эффект состаривания изображения (сепия). Значение 0 соответствует исходному изображению. Значение 1 или 100% полностью применяет эффект:

```
#img1       { filter: sepia(1); transition: 1s }
#img1:hover { filter: sepia(0) }
```

- invert(<Значение>) — инвертирует цвета изображения. Значение 0 соответствует исходному изображению. Значение 1 или 100% создает негатив:

```
#img1       { filter: invert(1); transition: 1s }
#img1:hover { filter: invert(0) }
```

2.19.5. Создание тени

Добавить тень можно с помощью функции drop-shadow(), атрибута text-shadow (тень для текста) и атрибута box-shadow (тень для блока).

Функция *drop-shadow()*

Функция `drop-shadow()` позволяет добавить тень. Формат функции:

```
drop-shadow(<X> <Y> [<Радиус размытия>] [<Цвет>])
```

В первых двух параметрах указывается смещение тени по осям `X` и `Y`. Третий параметр задает радиус размытия тени: чем больше значение, тем сильнее тень размывается. Если параметр не указан или равен нулю, то тень будет четкой без размытия. Четвертый параметр задает цвет тени. Если параметр не указан, то будет использоваться значение атрибута `color`.

Пример создания тени:

```
#img1 { filter: drop-shadow(3px 3px 5px rgba(0,0,0,0.7)) }
```

Создание тени для текста

Добавить тень для текста позволяет атрибут `text-shadow`. Его форматы:

```
text-shadow: <X> <Y> [<Радиус размытия>] [<Цвет>]
text-shadow: <Параметры тени 1>[, ..., <Параметры тени N>]
text-shadow: none
```

В первом формате в первых двух параметрах указывается смещение тени по осям `X` и `Y`. Третий параметр задает радиус размытия тени: чем больше значение, тем сильнее тень размывается. Если параметр не указан или равен нулю, то тень будет четкой без размытия. Четвертый параметр задает цвет тени. Если параметр не указан, то будет использоваться значение атрибута `color`.

Можно указывать несколько параметров тени через запятую.

Значение `none` убирает тень (поведение по умолчанию).

Здесь мы задали для заголовков второго уровня черную тень с уровнем прозрачности `0.2`, расположенную правее и ниже текста на `1 мм` и не имеющую эффекта размытия:

```
h2 { text-shadow: 1mm 1mm rgba(0, 0, 0, 0.2) }
```

Для заголовков первого уровня зададим красную тень, расположенную непосредственно под текстом и имеющую радиус размытия `10` пикселов:

```
h1 { text-shadow: 0mm 0mm 10px red }
```

Создание тени для блока

Добавить тень для блока позволяет атрибут `box-shadow`. Его форматы:

```
box-shadow: <X> <Y> [<Радиус размытия>] [<Расширение>] [<Цвет>] [inset]
box-shadow: <Параметры тени 1>[, ..., <Параметры тени N>]
box-shadow: none
```

Параметр <Расширение> задает величину, на которую тень увеличится относительно своих изначальных размеров (они равны размерам блочного элемента, для которого она создается). Если этот параметр не указан, расширение будет равно нулю.

Вот пример создания внешней тени:

```
h1 { box-shadow: 1px 1px 2px 1px rgba(0, 0, 0, 0.2); padding: 5px }
```

Если указать слово `inset`, будет создана внутренняя тень, визуально находящаяся внутри блока:

```
div { box-shadow: 0px 0px 10px 10px rgba(0,0,0,0.3) inset;
      padding: 15px; background: orange; height: 100px }
```

Назначение остальных параметров нам уже знакомо.

2.20. Анимация с двумя состояниями

Ранее, до появления CSS 3, для создания на страницах анимации, даже относительно простой, требовалось писать весьма сложные программы. Но сейчас задача Web-аниматоров существенно упростилась — чтобы заставить элемент страницы менять местоположение, размер или цвет, достаточно нескольких строчек CSS-кода.

Анимация в стиле CSS 3 формируется путем изменения значения указанного нами атрибута в течение заданного нами времени и по определенному нами закону.

Поддержка анимации с двумя состояниями появилась в браузерах Internet Explorer 10, Firefox 16.0, Chrome 26.0, Opera 12.1 и Safari 6.1.

Начальное состояние анимации задается стилем, который привязывается к анимируемому элементу страницы изначально. Конечное состояние записывается в другом стиле, который привязывается к элементу в момент начала анимации. Привязать второй стиль можно как программно, так и с помощью псевдокласса `:hover`, который применяет стиль при наведении курсора мыши на элемент:

```
<Селектор> {
  /* Задаем начальное состояние анимации */
}
<Селектор>:hover {
  /* Задаем конечное состояние анимации */
}
```

В следующем примере изначально изображение будет полупрозрачным. При наведении курсора мыши изображение плавно в течение одной секунды станет полностью непрозрачным. При выведении курсора изображение опять плавно станет полупрозрачным:

```
#img1       { opacity: .5; transition-duration: 1s }
#img1:hover { opacity: 1 }
```

2.20.1. Продолжительность анимации

Продолжительность анимации задает атрибут `transition-duration`:

```
transition-duration: <Время 1>[, ..., <Время N>]
```

Для указания значения времени мы можем использовать единицы измерения `s` (секунды) и `ms` (миллисекунды). Значение по умолчанию — `0` (т. е. анимации нет).

В следующем примере мы создали анимацию, запускающуюся, как только на элемент страницы наводится курсор мыши. Если мы уберем курсор с анимируемого элемента, тот мгновенно вернется в свое изначальное состояние без анимации:

```
#animated { background-color: #ff0000 }
#animated:hover { background-color: #00ff00; transition-duration: 3s }
```

Чтобы анимация проходила в обе стороны, нужно указать продолжительность и в начальном, и в конечном состояниях:

```
#animated { background-color: #ff0000; transition-duration: 1s }
#animated:hover { background-color: #00ff00; transition-duration: 3s }
```

При наведении курсора мыши цвет фона будет плавно меняться в течение трех секунд, а при выведении — плавно в обратную сторону в течение одной секунды.

Если продолжительность анимации в обе стороны должна быть одинаковой, то можно ее указать только в начальном состоянии:

```
#animated { background-color: #ff0000; transition-duration: 3s }
#animated:hover { background-color: #00ff00 }
```

Теперь цвет фона будет плавно меняться с красного на зеленый в течение трех секунд в обе стороны.

2.20.2. Задержка перед началом анимации

Задержку перед началом анимации задает атрибут `transition-delay`:

```
transition-delay: <Время 1>[, ..., <Время N>]
```

Для указания значения времени мы можем использовать единицы измерения `s` (секунды) и `ms` (миллисекунды). Значение по умолчанию — `0` (т. е. анимация начинается без задержки).

В следующем примере при наведении курсора мыши на элемент анимация начнется с задержкой в одну секунду, а при выведении курсора — сразу:

```
#animated { background-color: #ff0000; transition-duration: 3s;
            transition-delay: 0s }
#animated:hover { background-color: #00ff00;
                  transition-duration: 3s; transition-delay: 1s }
```

2.20.3. Задание анимируемых атрибутов

Задать атрибуты, значения которых будут меняться в процессе выполнения анимации (анимируемые атрибуты), позволяет атрибут `transition-property`:

```
transition-property: <Атрибут 1>[, ..., <Атрибут N>]
```

Названия атрибутов указываются без кавычек через запятую:

```
#animated { background-color: #ff0000; color: black;
            transition-duration: 3s; height: 100px;
            transition-property: background-color, color }
#animated:hover { background-color: #00ff00; color: red;
                  height: 200px }
```

В этом примере цвет текста и фона будет изменен с анимацией, а высота изменится сразу, т. к. атрибут `height` не указан в качестве значения атрибута `transition-property`.

Доступны также два особых значения:

- `all` — в анимацию будут вовлечены все атрибуты, значения которых изменились (поведение по умолчанию);
- `none` — отключает анимацию.

> ***Внимание!***
> Если задать значения атрибутов в процентах, то анимация выполнена не будет.

2.20.4. Закон анимации

Установить закон, по которому будет изменяться значение анимируемого атрибута, позволяет атрибут `transition-timing-function`:

```
transition-timing-function: <Закон 1>[, ..., <Закон N>]
```

CSS 3 поддерживает довольно много таких законов:

- `ease` — в начале скорость анимации слегка увеличивается, а в конце слегка уменьшается (поведение по умолчанию);
- `ease-in` — медленная скорость в начале и ускорение к концу анимации;
- `ease-out` — ускорение в начале и замедление к концу анимации;
- `ease-in-out` — медленная скорость в начале и замедление к концу анимации;
- `linear` — линейный закон, анимация будет выполняться с постоянной скоростью;
- `cubic-bezier(<Горизонтальная координата первой опорной точки>, <Вертикальная координата первой опорной точки>, <Горизонтальная координата второй опорной точки>, <Вертикальная координата второй опорной точки>)` — закон, соответствующий кубической кривой Безье.

  Значения координат ее опорных точек должны быть заданы в диапазоне между `0` и `1`, где `0` обозначает начало анимации, а `1` — ее конец. Однако для всех атрибутов, за исключением `color`, вертикальная координата может выходить за указанный диапазон, что позволяет достичь эффекта «эластичной» анимации;
- `steps(<Количество шагов>[, start | end])` — значение анимируемого атрибута изменяется не плавно, а скачкообразно, в начале или в конце одного из шагов

анимации, количество которых мы задали. Количество шагов анимации указывается в виде числа без единицы измерения. Если вторым значением указано `start`, изменение значения анимируемого атрибута выполняется в начале каждого шага, если `end` или если второе значение вообще не указано, — в его конце;

- `step-start` — анимируемый атрибут сразу получит свое конечное значение, которое в дальнейшем не изменится. Эквивалентно указанию `steps(1, start)`;
- `step-end` — анимируемый атрибут будет иметь свое начальное значение, которое в конце анимации мгновенно изменится на конечное. Эквивалентно указанию `steps(1, end)`.

Вот пример анимации, изменяющейся по линейному закону:

```
#animated { background-color: green; width: 100px;
            transition-duration: 3s;
            transition-timing-function: linear }
#animated:hover { width: 500px }
```

Задаем закон анимации, соответствующий кубической кривой Безье:

```
#animated { background-color: green; width: 100px;
            transition-duration: 3s;
            transition-timing-function: cubic-bezier(0.1,0.5,0.9,0.5) }
#animated:hover { width: 500px }
```

А процесс этой анимации будет разбит на пять шагов, и значение анимируемого атрибута будет скачкообразно меняться в начале каждого шага:

```
#animated { background-color: green; width: 100px;
            transition-duration: 3s;
            transition-timing-function: steps(5, start) }
#animated:hover { width: 500px }
```

2.20.5. Одновременное задание всех параметров анимации

Задать все параметры анимации одновременно позволяет атрибут `transition`:

```
transition: [<property>] <duration> [<timing-function>] [<delay>]
transition: <Анимация 1>[, ..., <Анимация N>]
```

Значение по умолчанию: `all 0s ease 0s`.

Пример:

```
#animated { background-color: green; width: 100px;
            transition: width 3s ease-in 0.5s }
#animated:hover { width: 500px }
```

Можно указать только продолжительность анимации:

```
#animated { background-color: green; width: 100px;
            transition: 3s }
#animated:hover { width: 500px }
```

2.20.6. Сложная анимация

Что, если нам понадобится изменять в процессе анимации значения отдельных атрибутов не так, как остальных? Для этого мы можем воспользоваться возможностями CSS по созданию сложной анимации.

Сначала укажем все анимируемые атрибуты в атрибуте `transition-property`, разделив их запятыми:

```
transition-property: color, background-color, font-size, width, height;
```

Потом зададим параметры анимации для всех этих атрибутов, указав их в атрибутах `transition-duration`, `transition-delay` и `transition-timing-function` также через запятую:

```
transition-duration: 3s, 2s, 3s, 1s, 1.5s;
```

Первое значение параметра будет относиться к первому анимируемому атрибуту, второе — ко второму и т. д. Так, в нашем случае значение цвета текста будет изменяться в течение трех секунд, значение цвета фона — в течение двух секунд, а значение размера шрифта — в течение трех секунд и т. д.

Если в атрибутах `transition-duration`, `transition-delay` или `transition-timing-function` указано меньше значений, чем атрибутов в `transition-property`, набор указанных значений будет повторяться столько раз, чтобы «покрыть» все указанные анимируемые атрибуты:

```
transition-property: color, background-color, font-size, width, height;
transition-duration: 3s, 2s;
```

В этом случае:

- значение атрибута `color` будет изменяться в течение трех секунд;
- значение атрибута `background-color` — в течение двух секунд;
- `font-size` — трех секунд;
- `width` — двух;
- `height` — трех.

Если же указать в атрибутах `transition-duration`, `transition-delay` или `transition-timing-function` большее число значений, лишние значения будут проигнорированы:

```
#animated { background-color: green; width: 100px; height: 100px;
  font-size: 10pt; color: black;
  transition-property: color, background-color, font-size, width, height;
  transition-duration: 3s, 2s, 3s, 1s, 1.5s
}
#animated:hover { width: 500px; height: 500px;
                  background-color: red; color: green; font-size: 30pt
}
```

2.21. Анимация с несколькими состояниями

Анимация с несколькими состояниями включает в свой состав несколько ключевых кадров, расположенных на шкале времени (*покадровая анимация*). Каждый ключевой кадр описывает набор анимируемых атрибутов и их значений, а шкала времени задает положение этих ключевых кадров во времени.

Полноценная поддержка анимации с несколькими состояниями появилась в браузерах Internet Explorer 10, Firefox 16.0, Chrome 43.0, Opera 30.0 и Safari 9.0. В более ранних версиях браузеров нужно использовать «вендорные» префиксы.

2.21.1. Шкала времени

Шкала времени описывается с помощью правила `@keyframes` по следующей схеме:

```
@keyframes <Название шкалы времени> {
   <Позиция 1>[, <Позиция 2>] {
      <Ключевой кадр 1>
   }
   <Позиция 3>[, <Позиция 4>] {
      <Ключевой кадр 2>
   }
   <Позиция N-1>[, <Позиция N>] {
      <Ключевой кадр N>
   }
}
```

Название шкалы времени указывается без кавычек и должно быть уникальным в пределах страницы. В нем допускается использовать буквы латиницы, цифры, дефисы и знаки подчеркивания, причем начинаться оно должно с буквы.

Положение ключевого кадра на шкале времени указывается в процентах от значения продолжительности анимации (как ее задать, мы узнаем позже). Значение `0%` задает начало анимации, `100%` — ее конец, а, скажем, `50%` — ее середину. Вместо `0%` можно использовать ключевое слово `from`, а вместо `100%` — `to`.

Каждый ключевой кадр описывает набор анимируемых атрибутов и их значений.

В следующем примере анимации с тремя ключевыми кадрами при наведении курсора мыши ширина блока плавно увеличится, а затем уменьшится. Продолжительность анимации составит две секунды. Название шкалы времени (`myAnimation`) указывается в качестве значения атрибута `animation`:

```
@keyframes myAnimation {
   0%   { width: 100px }
   50%  { width: 200px }
   100% { width: 100px }
}
#animated { background: green; width: 100px; height: 100px }
#animated:hover { animation: myAnimation 2s linear }
```

А здесь мы описываем шкалу времени из пяти кадров с именем `sample`:

```
@keyframes sample {
   from { left: 100px; top: 50px  }
   25%  { left: 200px; top: 50px  }
   50%  { left: 200px; top: 150px }
   75%  { left: 100px; top: 150px }
   to   { left: 100px; top: 50px  }
}
#animated { background: green; width: 100px; height: 100px;
            position: absolute; left: 100px; top: 50px;
            animation: sample 5s linear infinite }
```

В результате анимируемый элемент будет двигаться по траектории в виде квадрата бесконечное число раз. Продолжительность одной фазы анимации составит пять секунд.

2.21.2. Указание названия шкалы времени

Создав шкалу времени, мы можем задать ее для анимируемого элемента. Название шкалы времени (без кавычек) указывается в атрибуте `animation-name`:

```
animation-name: <Название шкалы времени 1>[, ...,
                <Название шкалы времени N>]
```

Пример:

```
animation-name: sample;
```

Можно указать сразу несколько названий через запятую:

```
animation-name: sample, myAnimation;
```

Предопределенное значение `none` отключает любую анимацию элемента (это, кстати, поведение по умолчанию).

2.21.3. Продолжительность анимации

Продолжительность анимации задает атрибут `animation-duration`:

```
animation-duration: <Время 1>[, ..., <Время N>]
```

Для указания значения времени мы можем использовать единицы измерения `s` (секунды) и `ms` (миллисекунды). Значение по умолчанию — `0` (т. е. анимации нет):

```
animation-duration: 5s;
```

2.21.4. Задержка перед началом анимации

Задержку перед началом анимации задает атрибут `animation-delay`:

```
animation-delay: <Время 1>[, ..., <Время N>]
```

Для указания значения времени мы можем использовать единицы измерения `s` (секунды) и `ms` (миллисекунды). Значение по умолчанию — `0` (т. е. анимация начинается без задержки):

```
animation-delay: 3s;
```

2.21.5. Закон анимации

Закон, по которому будет выполняться анимация, устанавливает атрибут `animation-timing-function`:

```
animation-timing-function: <Закон 1>[, ..., <Закон N>]
```

Можно указать те же самые законы, что и у атрибута `transition-timing-function` (см. *разд. 2.20.4*):

```
animation-timing-function: linear;
```

2.21.6. Количество повторений анимации

Количество повторений (проходов) анимации задает атрибут `animation-iteration-count`. Это количество указывается числом без единицы измерения. Значение `infinite` задает бесконечное повторение анимации. Значение по умолчанию — `1`.

Задаем пять повторений:

```
animation-iteration-count: 5;
```

А вот пример бесконечной анимации:

```
#animated { background: green; width: 100px; height: 100px;
            position: absolute; left: 100px; top: 50px;
            animation-name: sample;
            animation-duration: 5s;
            animation-delay: 0s;
            animation-timing-function: linear;
            animation-iteration-count: infinite }
```

2.21.7. Направление анимации

В каком направлении будет воспроизводиться анимация (в прямом, как мы задали на шкале времени, в обратном или то в прямом, то в обратном) позволяет указать атрибут `animation-direction`. Он может принимать четыре значения:

- `normal` — прямое направление анимации (значение по умолчанию);
- `reverse` — обратное направление анимации;
- `alternate` — нечетные проходы анимации воспроизводятся в прямом направлении, а четные — в обратном;
- `alternate-reverse` — нечетные проходы анимации воспроизводятся в обратном направлении, а четные — в прямом.

Понятно, что значения alternate и alternate-reverse имеет смысл указывать лишь в тех случаях, если анимация повторяется более одного раза:

```
@keyframes myAnimation {
   0%   { left: 100px }
   100% { left: 500px  }
}
#animated { background: green; width: 100px; height: 100px;
            position: absolute; left: 100px; top: 50px;
            animation-name: myAnimation;
            animation-duration: 2s;
            animation-timing-function: ease-in-out;
            animation-iteration-count: infinite;
            animation-direction: alternate }
```

2.21.8. Текущее состояние анимации

Текущее состояние анимации: анимация воспроизводится (значение running, используемое по умолчанию) или приостановлена (значение paused) — задает атрибут animation-play-state.

Создадим анимацию, которая начинается сразу после загрузки страницы и приостанавливается, если навести курсор мыши на элемент:

```
#animated { background: green; width: 100px; height: 100px;
            position: absolute; left: 100px; top: 50px;
            animation-name: myAnimation;
            animation-duration: 2s;
            animation-timing-function: ease-in-out;
            animation-iteration-count: infinite;
            animation-direction: alternate;
            animation-play-state: running }
#animated:hover { animation-play-state: paused }
```

2.21.9. Состояние элемента до начала анимации и после ее завершения

Состояние элемента до начала анимации и после ее завершения определяет атрибут animation-fill-mode. Он может принимать четыре значения:

- none — элемент окажется в состоянии, заданном в привязанном к нему первом стиле. Состояния, описанные в ключевых кадрах анимации, в этом случае в расчет не принимаются. Это значение по умолчанию;
- backwards — до начала анимации, на период действия задержки (значение атрибута animation-delay), элемент окажется в состоянии, описанном в начальном ключевом кадре (это может быть первый ключевой кадр или последний — в зависимости от значения атрибута animation-direction). По умолчанию на период

задержки элемент остается в состоянии, заданном в привязанном к нему первом стиле, и не зависит от ключевых кадров;

- `forwards` — после завершения анимации элемент окажется в состоянии, описанном в конечном ключевом кадре (это может быть последний ключевой кадр или первый — в зависимости от значений атрибутов `animation-direction` и `animation-iteration-count`);
- `both` — комбинация значений `backwards` и `forwards`.

В следующем примере изначально элемент расположен в положении `100 px` от левого края, а анимация начинается от положения `0 px`. Так как задано значение `both`, после запуска анимации на период задержки (три секунды) элемент окажется в положении `0 px` и будет там находиться до начала анимации. Если бы мы не указали значение `both` или `backwards`, то он находился бы в позиции `100 px`. После завершения анимации элемент останется в положении `500 px`, которое задано последним ключевым кадром. Если бы мы не указали значение `both` или `forwards`, то он находился бы в позиции `100 px`:

```
@keyframes myAnimation {
   0%   { left: 0 }
   100% { left: 500px }
}
#animated { background: green; width: 100px; height: 100px;
            position: absolute; left: 100px; top: 50px;
            animation-name: myAnimation;
            animation-duration: 2s;
            animation-iteration-count: 1;
            animation-direction: normal;
            animation-delay: 3s;
            animation-fill-mode: both }
```

2.21.10. Одновременное задание всех параметров анимации

Задать все параметры анимации сразу позволяет атрибут `animation`:

```
animation: <animation-name> <animation-duration>
           [<animation-timing-function>] [<animation-delay>]
           [<animation-iteration-count>] [<animation-direction>]
           [<animation-fill-mode>] [<animation-play-state>]
```

Вот пример его использования:

```
#animated { background: green; width: 100px; height: 100px;
    position: absolute; left: 100px; top: 50px;
    animation: sample 5s linear infinite alternate }
```

2.21.11. Сложная анимация

И, наконец, как и в случае анимации с двумя состояниями, мы можем создать более сложную анимацию, указав параметры через запятую:

```
@keyframes anim1 {
   0%   { left: 0 }
   100% { left: 500px }
}
@keyframes anim2 {
   0%   { background-color: red }
   100% { background-color: blue }
}
#animated { background: green; width: 100px; height: 100px;
            position: absolute; left: 100px; top: 50px;
            animation-name: anim1, anim2;
            animation-duration: 6s, 3s;
            animation-iteration-count: 1, 2 }
```

Здесь действуют те же правила, что мы изучили применительно к анимации с двумя состояниями.

2.22. Двумерные трансформации

CSS 3 позволяет выполнять различные трансформации с элементами: смещать их относительно изначального местоположения, наклонять, вращать и изменять их масштаб. Такие манипуляции можно выполнить без необходимости превращения элемента в свободно позиционированный.

Трансформации бывают *двумерными* и *трехмерными*. Мы начнем с двумерных трансформаций, которые выполняются в плоскости страницы.

Поддержка трансформаций появилась в браузерах Internet Explorer 10, Firefox 16.0, Chrome 36.0, Opera 23.0 и Safari 9.0. Для предыдущих версий браузеров нужно использовать «вендорные» префиксы.

2.22.1. Атрибут *transform*

Способ трансформации задается с помощью атрибута `transform`:

```
transform: <Функция 1>[ ... <Функция N>]
transform: none
```

В качестве значения указывается функция, задающая трансформацию. Можно указать несколько функций через пробел. Параметры трансформации задаются в виде аргументов функции (указываются через запятую) в круглых скобках, которые ставятся сразу же после ее имени:

```
#div1 { background: red; width: 100px; height: 100px;
        transform: translate(100px, 200px) }
```

```
#div2 { background: green; width: 100px; height: 100px;
        position: absolute; left: 200px; top: 50px;
        transform: rotate(45deg) }
...
<div id="div1">div1</div>
<div id="div2">div2</div>
```

В этом примере элемент с идентификатором `div1` сместится на `100` пикселов вправо и `200` пикселов вниз относительно начального положения, а элемент с идентификатором `div2` будет повернут на `45` градусов.

В дальнейшем разговор в основном пойдет о функциях, применяемых для указания трансформаций и их параметров. Попутно мы рассмотрим еще несколько атрибутов, которые могут нам пригодиться.

2.22.2. Смещение

Для смещения элемента относительно начального положения предназначены следующие функции:

- `translateX(<tx>)` — выполняет смещение элемента по горизонтали. Положительные значения смещают элемент вправо, а отрицательные — влево. Сдвинем элемент на `100` пикселов вправо:

  ```
  transform: translateX(100px);
  ```

- `translateY(<ty>)` — выполняет смещение элемента по вертикали. Положительные значения смещают элемент вниз, а отрицательные — вверх. Сдвинем элемент на `50` пикселов вверх:

  ```
  transform: translateY(-50px);
  ```

- `translate(<tx>[, <ty>])` — позволяет сместить элемент сразу по обеим координатным осям. В качестве первого параметра указывается величина смещения по горизонтали, а в качестве второго — величина смещения по вертикали. Сдвинем элемент на `50` пикселов влево и `100` пикселов вниз:

  ```
  transform: translate(-50px, 100px);
  ```

2.22.3. Изменение масштаба

Изменить масштаб элемента позволяют следующие функции:

- `scaleX(<sx>)` — выполняет масштабирование вдоль горизонтальной оси координат. В качестве параметра указывается относительная величина, на которую следует изменить масштаб. Значение больше единицы задает увеличение масштаба, а значение меньше единицы — его уменьшение. Единица измерения при этом не указывается.

  Вот пример увеличения масштаба по горизонтали в два раза:

  ```
  transform: scaleX(2);
  ```

Если указать значение `-1`, то элемент будет зеркально отображен по горизонтали:

```
transform: scaleX(-1);
```

- `scaleY(<sy>)` — выполняет масштабирование вдоль вертикальной оси координат.

  Вот пример уменьшения масштаба по вертикали в два раза:

  ```
  transform: scaleY(0.5);
  ```

  Если указать значение `-1`, то элемент будет зеркально отображен по вертикали:

  ```
  transform: scaleY(-1);
  ```

- `scale(<sx>[, <sy>])` — выполняет масштабирование элемента одновременно по обеим координатным осям. Первый параметр функции задает величину масштаба по горизонтали, а второй — по вертикали. Если указан только один параметр, то изменение масштаба будет выполнено пропорционально по обеим координатным осям. Растянем контейнер вдвое в ширину и сожмем вдвое в высоту:

  ```
  transform: scale(2, 0.5);
  ```

По умолчанию точка, относительно которой выполняется изменение масштаба, расположена в центре элемента. С помощью атрибута `transform-origin` (см. *разд. 2.22.7*) мы можем сместить ее в другое место, например в левый верхний угол:

```
#div1 { background: red; width: 100px; height: 100px;
        transform-origin: left top; transform: scale(2) }
```

2.22.4. Наклон

Наклонить элемент страницы позволяют следующие функции:

- `skewX(<ax>)` — выполняет наклон по горизонтальной оси координат. В качестве параметра указывается угол, на который требуется выполнить наклон. Угол этот отсчитывается против часовой стрелки. Чтобы выполнить наклон по часовой стрелке, следует указать отрицательное значение угла. Наклоняем контейнер на 25° по горизонтали против часовой стрелки:

  ```
  transform: skewX(25deg);
  ```

- `skewY(<ay>)` — выполняет наклон по вертикальной оси координат. Наклоняем контейнер на 5° по вертикали по часовой стрелке:

  ```
  transform: skewY(-5deg);
  ```

- `skew(<ax>[, <ay>])` — наклоняет элемент сразу по горизонтали и вертикали. В качестве первого параметра указывается угол наклона по горизонтали, а в качестве второго — угол наклона по вертикали:

  ```
  transform: skew(25deg, -5deg);
  ```

Элемент будет наклоняться относительно точки начала координат. По умолчанию она находится в середине элемента. С помощью атрибута `transform-origin` (см. *разд. 2.22.7*) ее можно сместить в другое место:

```
#div1 { background: red; width: 100px; height: 100px;
        transform-origin: left top; transform: skew(25deg) }
```

2.22.5. Вращение

Повернуть элемент на заданный угол позволяет функция `rotate(<Угол>)`. Нужный угол задается в качестве ее единственного параметра. Положительное значение угла задает поворот по часовой стрелке, отрицательное — против часовой стрелки:

- поворачиваем элемент на `45°` по часовой стрелке:

  ```
  transform: rotate(45deg);
  ```

- поворачиваем элемент на 90° против часовой стрелки:

  ```
  transform: rotate(-90deg);
  ```

  Если этот элемент содержит текст, мы получим вертикальную надпись.

Точка начала координат, вокруг которой поворачивается элемент, по умолчанию находится в его центре. С помощью атрибута `transform-origin` (см. *разд. 2.22.7*) ее можно сместить в другое место:

```
#div1 { background: red; width: 100px; height: 100px;
        transform-origin: left top; transform: rotate(-20deg) }
```

2.22.6. Применение матрицы трансформации

Применить произвольную матрицу трансформации позволяет функция `matrix()`:

```
matrix(mxx, mxy, myx, myy, tx, ty)
```

Матрица трансформации имеет следующий вид:

```
mxx  mxy  tx
myx  myy  ty
```

Вычисление координат с учетом трансформации производится так:

```
x = mxx * x + mxy * y + tx
y = myx * x + myy * y + ty
```

Вот пример матрицы трансформации для масштабирования в два раза:

```
transform: matrix(2, 0, 0, 2, 0, 0);
```

Матрица трансформации для вращения имеет следующий вид:

```
 cos(a)  sin(a)  tx
-sin(a)  cos(a)  ty
```

Вот пример вращения элемента на угол `45°` по часовой стрелке (синус и косинус этого угла равен `0.7071`):

```
transform: matrix(0.7071, 0.7071, -0.7071, 0.7071, 0, 0);
```

2.22.7. Позиционирование точки начала координат для двумерных трансформаций

Ранее неоднократно отмечалось, что по умолчанию точка начала координат располагается в середине элемента страницы, к которому применяются трансформации. Соответственно, относительно этой точки элемент будет изменять масштаб, наклоняться и вращаться. Но у нас есть возможность установить точку начала координат в любое место элемента с помощью атрибута `transform-origin`:

```
transform-origin: <Горизонтальная координата> <Вертикальная координата>
```

Горизонтальную координату можно указать в виде числа, представляющего собой расстояние от левой границы элемента до самой точки или до одного из предопределенных значений: `left` (левая граница элемента), `center` (центр) или `right` (правая граница). Вертикальная координата также задается в виде числового расстояния либо от верхней границы элемента, либо от одного из предопределенных значений: `top` (верхняя граница элемента), `center` (центр) и `bottom` (нижняя граница).

Значение по умолчанию: `50% 50%` (т. е. точка начала координат находится в центре элемента).

Указав местоположение точки начала координат в абсолютных единицах измерения, мы можем даже вынести эту точку за пределы элемента. Иногда такой прием может быть полезным.

В следующем примере мы смещаем точку начала координат в верхний левый угол блока и поворачиваем его на `45°` по часовой стрелке:

```
#div2 { background: green; width: 100px; height: 100px;
        position: absolute; left: 200px; top: 50px;
        transform-origin: left top; transform: rotate(45deg) }
```

В результате блок будет повернут вокруг своего левого верхнего угла.

2.22.8. Сложные двумерные трансформации

Как сместить, изменить масштаб, наклонить или повернуть элемент страницы, мы выяснили. Но что, если нам потребуется, скажем, одновременно сместить и повернуть его, т. е. выполнить сложное преобразование? Для этого достаточно все функции, указывающие отдельные преобразования, записать в значении атрибута `transform` друг за другом, разделив их пробелами:

- смещаем элемент на `200` пикселов вниз и увеличиваем его ширину вдвое:

  ```
  transform: translateY(200px) scaleX(2);
  ```

- наклоняем элемент на 15° против часовой стрелки и поворачиваем на 45° также против часовой стрелки:

  ```
  transform: skewX(15deg) rotate(-45deg);
  ```

2.23. Трехмерные трансформации

Трехмерные трансформации выполняются в воображаемом трехмерном пространстве, проецируемом на плоскость страницы.

Двумерные трансформации выполняются по горизонтальной и вертикальной координатным осям, носящим в математике наименования `X` и `Y`. В случае трехмерных преобразований к ним добавляется ось `Z`, направленная из точки начала координат перпендикулярно воображаемой поверхности элемента. Положительные значения координаты по этой оси отсчитываются в направлении к посетителю, отрицательные — от посетителя.

Все трехмерные трансформации элементов страницы выполняются в ортогональной проекции на поверхность контейнера, в котором они находятся.

2.23.1. Перспектива

Перспектива (ее также называют *глубиной перспективы*) в CSS 3 — это отсчитываемое по оси `Z` расстояние между воображаемым наблюдателем (посетителем) и поверхностью контейнера, в котором находится подвергаемый трансформациям элемент. Фактически перспектива добавляет трехмерному изображению «глубину», благодаря чему мы получим на странице тот эффект, которого собираемся добиться, применяя трехмерные трансформации.

По умолчанию перспектива равна нулю, т. е. фактически отсутствует. Из-за этого при выполнении трехмерных трансформаций мы получим не тот результат, на который рассчитываем.

Следовательно, сначала нам нужно указать перспективу. Сделать это можно с помощью атрибута `perspective`, указав в качестве его значения величину перспективы. Значение `none` убирает перспективу, делая ее равной нулю (значение по умолчанию).

Запомним, что этот атрибут и, соответственно, перспектива указывается для контейнера элемента, а не для самого элемента.

Задаем перспективу в `300` пикселов для всех элементов, находящихся на странице:

```
body { perspective: 300px }
```

Функция `perspective()` позволяет задать перспективу для самого элемента:

```
#img1 { transform: perspective(900px) rotateX(45deg) }
...
<img alt="" src="photo.jpg" id="img1">
```

2.23.2. Выполнение трехмерных трансформаций

В случае трехмерных трансформаций для смещения, масштабирования и наклона элементов по горизонтальной и вертикальной координатным осям мы можем использовать те же функции, что применяли для выполнения двумерных трансфор-

маций. Так что здесь нам остается лишь изучить функции, специфичные для собственно трехмерных трансформаций.

Вот поддерживаемые функции, задаваемые ими трехмерные трансформации и примеры их использования:

- translateZ(<tz>) — смещение элемента по оси Z. Единственный параметр задает величину смещения. Положительные значения координат отсчитываются в направлении к посетителю, отрицательные — от посетителя:

  ```
  transform: translateZ(50px);
  transform: translateZ(-50px);
  ```

- translate3d(<tx>, <ty>, <tz>) — смещение элемента одновременно по всем координатным осям. Принимает три параметра, задающие, соответственно, смещения по осям X, Y и Z:

  ```
  transform: translate3d(100px, -200px, 1cm);
  ```

- scaleZ(<sz>) — масштабирование по оси Z. Единственный параметр задает масштаб элемента:

  ```
  transform: scaleZ(0.5) rotateX(45deg);
  ```

  Масштабировать элемент по оси Z имеет смысл только в том случае, если перед масштабированием к нему был применен поворот вокруг оси X или Y. В противном случае визуально этот элемент никак не изменится;

- scale3d(<sx>, <sy>, <sz>) — масштабирование сразу по всем трем координатным осям. Принимает три параметра, задающие, соответственно, величины масштаба по осям X, Y и Z:

  ```
  transform: scale3d(1.5, 0.5, 2);
  ```

- rotateX(<Угол>), rotateY(<Угол>) и rotateZ(<Угол>) — поворот элемента по координатным осям X, Y и Z соответственно. Единственный параметр задает угол поворота:

  ```
  transform: rotateX(45deg);
  transform: rotateY(-15deg);
  transform: rotateZ(0.5turn);
  ```

- rotate3d(<X>, <Y>, <Z>, <Угол>) — позволяет указать оси, вокруг которых выполняется вращение.

  Вот пример вращения вокруг оси Y на угол 60° по часовой стрелке:

  ```
  transform: rotate3d(0, 1, 0, 60deg);
  ```

2.23.3. Задание точки зрения

Мы уже знаем, что перед выполнением трехмерных трансформаций необходимо указать перспективу. В противном случае мы можем вообще не получить никакого видимого результата. Также мы знаем, что перспектива в CSS 3 — это расстояние

между наблюдателем и плоскостью контейнера элемента, над которым выполняется трансформация.

Но где же находится этот воображаемый наблюдатель? В особой точке, носящей название *точки зрения*. Ее Web-браузер и имеет в виду, когда рассчитывает проекцию трансформируемого элемента на двумерную плоскость его контейнера.

По умолчанию точка зрения находится в центре контейнера элемента, к которому применяется трансформация, и отстоит от него на величину перспективы по оси Z. И, что очень важно, всегда остается в этом месте независимо от того, какое преобразование мы применяем к элементам, находящимся в этом контейнере.

Но при необходимости мы можем сместить точку зрения в другое место, тем самым заставив воображаемого наблюдателя «взглянуть» на подвергаемый трансформациям элемент с другой стороны. Это можно сделать с помощью атрибута `perspective-origin`:

```
perspective-origin: <X> [<Y>]
```

Координаты проекции точки зрения указываются в том же формате, как и местоположение точки начала координат (атрибут `transform-origin`). Как видим, здесь указываются лишь горизонтальная и вертикальная координаты (координата точки зрения по оси Z — это фактически перспектива, задаваемая атрибутом стиля `perspective`). Если вертикальная координата не указана, она получит значение `50%`.

Значение атрибута `perspective-origin` по умолчанию: `50% 50%` (т. е. проекция точки зрения находится в центре контейнера).

Атрибут `perspective-origin`, как и атрибут `perspective`, указывается для контейнера элементов, к которым будут применены трехмерные трансформации.

В следующем примере мы задаем перспективу, местоположение точки зрения и трехмерное преобразование, которое будет выполнено с учетом нового расположения точки зрения:

```
body { perspective: 300px; perspective-origin: right top }
#img1 { transform: rotateY(175deg) }
```

2.23.4. Скрытие обратной стороны элемента

Рассмотрим код листинга 2.15.

Листинг 2.15. Пример трансформации элемента

```
body { perspective: 300px }
#transformed { font-size: 36pt; width: 150px; background-color: black;
               color: white; transform: rotateY(175deg) }
...
<div id="transformed">Блок 1</div>
```

Он создает контейнер с идентификатором `transformed` и поворачивает его относительно вертикальной оси на `175°` по часовой стрелке. В результате контейнер ока-



Рис. 2.5. Перевернутый элемент страницы с видимой «обратной» стороной

жется, если так можно сказать, перевернутым, и мы увидим его «обратную» сторону (рис. 2.5).

В некоторых случаях, например при создании с помощью трансформации трехмерных фигур, подобный результат неприемлем. Поэтому нам может потребоваться скрыть «обратную» сторону таких перевернутых элементов.

Для этого мы воспользуемся атрибутом `backface-visibility`. Он поддерживает два значения:

- `visible` — сделать «обратную» сторону элемента видимой (значение по умолчанию);
- `hidden` — полностью скрыть элемент, если он окажется повернутым к посетителю своей «обратной» стороной.

Атрибут `backface-visibility` указывается непосредственно для элемента, к которому применяются трехмерные трансформации:

```
#transformed { font-size: 36pt; width: 150px; background-color: black;
               color: white; transform: rotateY(175deg);
               backface-visibility: hidden }
```

Теперь Web-браузер вообще не выведет на экран контейнер `transformed`. Что вполне понятно — ведь этот элемент оказался повернутым к наблюдателю своей «обратной» стороной.

2.23.5. Режим проецирования элементов на контейнер

По умолчанию элементы, находящиеся в контейнере, к которому были применены трехмерные трансформации, отображаются в его плоскости. Если мы применим трехмерные трансформации и к ним, они так и останутся проекциями на плоскость контейнера.

Рассмотрим пример кода, приведенный в листинге 2.16.

Листинг 2.16. Трехмерные трансформации вложенных элементов

```
body { perspective: 300px }
#cont { width: 250px; height: 250px; background-color: yellow;
        transform: rotateX(45deg); margin: 20px 0 0 150px }
#ch { width: 150px; height: 150px; background-color: red;
      transform: translate3d(50px, 50px, 100px) }
...
<div id="cont">
  <div id="ch"></div>
</div>
```

Здесь мы создали два вложенных друг в друга контейнера и задали для них трехмерные трансформации: «внешний» контейнер повернули, а вложенный в него сместили, в том числе и по оси Z, чтобы визуально приподнять его над «внешним».

Результат, который покажет нам Web-браузер, представлен на рис. 2.6. Видно, что вложенный контейнер так и находится в плоскости «внешнего».

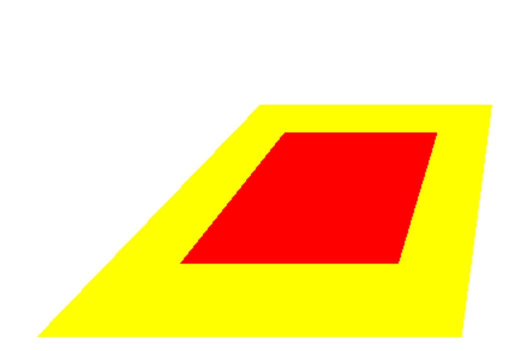


Рис. 2.6. Вложенный контейнер находится в плоскости «внешнего»

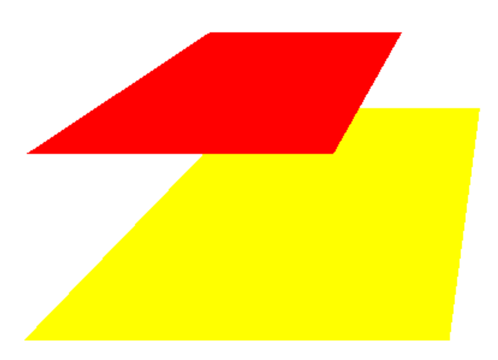


Рис. 2.7. Вложенный контейнер вышел из плоскости «внешнего»

Однако мы можем сделать так, чтобы вложенные элементы в случае применения к ним трехмерных трансформаций вышли бы из плоскости контейнера и отображались в виде проекции непосредственно на странице. Так можно создавать трехмерные фигуры.

Режим проецирования элементов на контейнер, в который они вложены, задается с помощью атрибута `transform-style`, поддерживающего два значения:

- `flat` — элементы выводятся в виде проекции на поверхность контейнера (значение по умолчанию);
- `preserve-3d` — элементы существуют в воображаемом трехмерном пространстве отдельно от контейнера и отображаются как проекции непосредственно на поверхности страницы.

Если мы добавим к приведенному в листинге 2.16 CSS-коду стиль:

```
#cont { transform-style: preserve-3d }
```

то Web-браузер выведет нам изображение, показанное на рис. 2.7. Мы видим, что вложенный контейнер вышел за пределы «внешнего» и существует как бы отдельно от него.

2.23.6. Позиционирование точки начала координат для трехмерных трансформаций

Для трехмерных трансформаций предусмотрен расширенный формат атрибута `transform-origin`, указывающего местоположение точки начала координат:

```
transform-origin: <X> <Y> <Z>
```

По умолчанию значение координаты по оси `Z` — `0`.

Пример:

```
transform: rotateX(0.2rad); transform-origin: left top 200px;
```

2.23.7. Сложные трехмерные трансформации

Для создания сложных трехмерных трансформаций мы можем пользоваться уже знакомым нам приемом, указывая функции, задающие отдельные трансформации, через пробел:

```
transform: translateZ(-200px) rotateY(30deg);
```

Мы также можем смешивать двумерные и трехмерные преобразования:

```
transform: skewX(15deg) rotateZ(-45deg);
```

2.24. Медиазапросы и адаптивный дизайн

По умолчанию стили применяются ко всем устройствам. Однако существует возможность указать разные таблицы стилей для разных устройств. Стили создаются для следующих типов устройств:

- `all` — для любых устройств (значение по умолчанию);
- `screen` — для экрана монитора;
- `print` — для предварительного просмотра и распечатки документа;
- `speech` — программы для чтения текста вслух (например, речевые браузеры).

Тип устройства можно указать несколькими способами. При подключении внешнего файла со стилями используется параметр `media="<Медиазапрос>"` тега `<link>`:

```
<link rel="stylesheet" type="text/css" media="all" href="all.css">
<link rel="stylesheet" type="text/css" media="print" href="print.css">
<link rel="stylesheet" type="text/css" media="screen" href="screen.css">
```

Подключить внешний CSS-файл можно также с помощью правила `@import`:

```
@import url(<URL-адрес>)[ <Медиазапрос>];
@import <URL-адрес>[ <Медиазапрос>];
```

Правило `@import` должно быть расположено внутри тега `<style>`:

```
<style type="text/css">
  @import "print.css" print;
</style>
```

Внутри CSS-кода используется правило `@media`:

```
@media <Медиазапрос> {
   /* Стили для указанного устройства */
}
```

Вот пример указания цвета фона документа для экрана монитора:

```
@media screen {
   body { background: green }
}
```

В параметре `<Медиазапрос>` можно указывать не только тип устройства, но и целые логические выражения. Стили будут применены только в том случае, если логическое выражение истинно. В логических выражениях используются следующие операторы:

- `and` — логическое `И`. Если тип устройства — экран монитора и его ширина меньше или равна `600` пикселам, то фон страницы будет красным, а в противном случае — зеленым:

  ```
  body { background: green }
  @media screen and (max-width: 600px) {
     body { background: red }
  }
  ```

- `not` — логическое отрицание. Позволяет изменить результат логического выражения на противоположный:

  ```
  body { background: green }
  @media not screen and (max-width: 600px) {
     body { background: red }
  }
  ```

  В этом примере при ширине экрана меньше или равной `600` пикселам фон страницы будет зеленым, а в противном случае — красным. Приоритет оператора `not` меньше приоритета оператора `and`, поэтому вначале будет вычислено выражение с оператором `and`, а лишь затем с оператором `not`;

- `запятая` — логическая операция `ИЛИ`. Если ширина экрана меньше (или равна) `600` пикселов или больше (или равна) `800` пикселов, то фон страницы будет красным, а в противном случае — зеленым:

  ```
  body { background: green }
  @media screen and (max-width: 600px),
         screen and (min-width: 800px) {
     body { background: red }
  }
  ```

Можно указывать сразу несколько логических выражений в одной инструкции. Если ширина экрана больше (или равна) `600` пикселов и меньше (или равна) `800` пикселов, то фон страницы будет красным, а в противном случае — зеленым:

```
body { background: green }
@media screen and (min-width: 600px) and (max-width: 800px) {
   body { background: red }
}
```

Перед типом устройства допускается указание необязательного ключевого слова `only`:

```
@media only screen { }
```

Эта инструкция эквивалента следующей:

```
@media screen { }
```

Характеристики устройства задаются с помощью следующих атрибутов:

- `width`, `min-width` и `max-width` — ширина области просмотра. Цвет фона страницы будет красным, если ширина равна `600` пикселам, в противном случае — зеленым:

  ```
  body { background: green }
  @media screen and (width: 600px) {
     body { background: red }
  }
  ```

- `height`, `min-height` и `max-height` — высота области просмотра;
- `aspect-ratio`, `min-aspect-ratio` и `max-aspect-ratio` — соотношение сторон. Значения указываются через символ `/`:

  ```
  @media screen and (aspect-ratio: 16/9) { }
  ```

- `orientation` — ориентация: `landscape` (альбомная) или `portrait` (портретная):

  ```
  @media screen and (orientation: landscape) { }
  ```

- `resolution`, `min-resolution` и `max-resolution` — разрешение устройства. Значение указывается в единицах измерения `dpi` (точки на дюйм) или `dpcm` (точки на сантиметр):

  ```
  @media print and (resolution: 300dpi) { }
  ```

- `color`, `min-color` и `max-color` — число битов на канал цвета:

  ```
  @media screen and (color: 8) { }
  ```

  Если значение не указано, то проверяется, что устройство поддерживает цвет:

  ```
  @media screen and (color) { }
  ```

- `color-index`, `min-color-index` и `max-color-index` — количество цветов в таблице или нуль, если устройство не использует таблицу цветов:

  ```
  @media screen and (color-index: 0) { }
  ```

- `monochrome`, `min-monochrome` и `max-monochrome` — определяют, является ли устройство монохромным:

  ```
  @media print and (monochrome) { }
  ```

Используя медиазапросы, мы можем создать адаптивный дизайн страницы. Такая страница будет подстраивать содержимое под различную ширину экрана и учитывать другие характеристики устройства, например ориентацию экрана смартфона.

Помните, что мобильный Интернет стал доступным всем, поэтому не забывайте про владельцев смартфонов и планшетов, т. к. их становится все больше и больше.

Web-браузер Firefox содержит инструмент, который позволяет наглядно увидеть как отображается содержимое страницы при различных характеристиках устройства. Перейдите в главное меню и выберите пункт **Разработка | Адаптивный дизайн** или нажмите комбинацию клавиш <Ctrl>+<Shift>+<M>. В окне Web-браузера отобразится содержимое страницы внутри рамки. Размер этой рамки можно задать, введя ширину и высоту в поля ввода, расположенные над рамкой, или взявшись мышью за правую или нижнюю границу. В заголовке рамки содержится список доступных устройств с уже настроенными характеристиками, а также значок, с помощью которого можно изменить ориентацию экрана.

> ***Примечание***
>
> Медиазапросы поддерживают также теги `<img>` (см. *разд. 1.6.1*) и `<picture>` (см. *разд. 1.6.3*).

2.25. Проверка CSS-кода на соответствие стандартам

После создания документа, содержащего каскадные таблицы стилей, его необходимо проверить на отсутствие ошибок и соответствие стандартам. Ведь можно случайно допустить опечатку в названии атрибута или указать некорректное значение. Web-браузеры обычно не сообщают об ошибках, а пытаются их обработать. Поэтому о существовании ошибок можно узнать только в случае, если Web-браузер неправильно их обработал, и это видно в его окне.

Для проверки CSS-кода предназначен сайт **http://jigsaw.w3.org/css-validator/**. Чтобы проверить документ, размещенный в Интернете, достаточно ввести URL-адрес и нажать кнопку **Проверить**. Можно также загрузить файл или вставить CSS-код в поле ввода многострочного текста. Если после проверки были обнаружены ошибки, то будет выведено их подробное описание. После исправления ошибок следует повторно проверить CSS-код.

> ***Внимание!***
>
> Если CSS-код встроен в HTML-документ, то сначала нужно проверить сам документ на отсутствие ошибок (см. *разд. 1.16*).



ГЛАВА 3

Основы JavaScript. Создаем страницы, реагирующие на действия пользователей

3.1. Первые шаги

JavaScript — это язык программирования, позволяющий сделать Web-страницу интерактивной, т. е. реагирующей на действия пользователя.

Последовательность инструкций (называемая *программой*, *скриптом* или *сценарием*) выполняется *интерпретатором*, встроенным в Web-браузер. Иными словами, код программы внедряется в HTML-документ и выполняется на стороне клиента. Для выполнения программы даже не нужно перезагружать Web-страницу. Все программы выполняются в результате возникновения какого-то события. Например, перед отправкой данных формы можно проверить их на допустимые значения и, если значения не соответствуют ожидаемым, запретить отправку данных.

3.1.1. Первая программа на JavaScript

При изучении языков программирования принято начинать с программы, выводящей надпись «Hello, world». Не будем нарушать традицию и продемонстрируем, как это будет выглядеть на JavaScript (листинг 3.1).

Листинг 3.1. Первая программа

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Первая программа</title>
</head>
<body>
<script>
document.write("Hello, world");
</script>
```

```
<noscript>
   <p>Ваш Web-браузер не поддерживает JavaScript</p>
</noscript>
</body>
</html>
```

Открываем редактор Notepad++ и создаем новый документ. В меню **Кодировки** устанавливаем флажок **Кодировать в UTF-8 (без BOM)**. Набираем код и сохраняем в формате HTML, например, под именем test.html. Запускаем Web-браузер и открываем сохраненный файл.

Возможны следующие варианты:

- в окне Web-браузера отображена надпись **Hello, world** — значит, все нормально;
- отобразилась надпись **Ваш Web-браузер не поддерживает JavaScript** и Web-браузер задает вопрос **Запустить скрипты?** — значит, в настройках Web-браузера установлен флажок **Подтверждать запуск скриптов**. Можно либо установить флажок **Разрешить запуск скриптов**, либо каждый раз отвечать **Да** на этот вопрос;
- отобразилась надпись **Ваш Web-браузер не поддерживает JavaScript** и Web-браузер не задает никаких вопросов — значит, в настройках Web-браузера установлен флажок **Запретить запуск скриптов**. Надо установить флажок **Разрешить запуск скриптов**;
- в окне Web-браузера нет никаких надписей — значит, допущена опечатка в коде программы. Следует иметь в виду, что в JavaScript регистр имеет важное значение — строчные и прописные буквы считаются разными. Более того, каждая буква, каждая кавычка имеет значение. Достаточно ошибиться в одной букве, и вся программа работать не будет.

Итак, мы столкнулись с первой проблемой при использовании JavaScript — любой пользователь может отключить запуск скриптов в настройках Web-браузера. Но эта проблема не единственная. Разные Web-браузеры могут по-разному выполнять код программы. По этой причине приходится писать персональный код под каждый Web-браузер или пользоваться JavaScript-библиотеками — например, jQuery. Все примеры скриптов в этой книге написаны под браузер Firefox версии 56 и могут не работать в других Web-браузерах. Увы, но просто использовать Web-браузер Firefox не достаточно — нужно еще учитывать его версию. Если у вас установлена более новая версия, то все должно работать, но в более ранних версиях некоторые возможности окажутся недоступны. Это следует помнить.

3.1.2. Тег *<script>*

Вернемся к нашему примеру. Программа внедряется в HTML-документ с помощью парного тега `<script>`. Тег имеет следующие параметры:

- `type` — задает MIME-тип содержимого. Для JavaScript указывается значение `text/javascript`. В HTML 5 это значение используется по умолчанию, поэтому

его можно не указывать, но в предыдущих версиях нужно указывать обязательно:

```
<script type="text/javascript">
document.write("Hello, world");
</script>
```

- src — позволяет указать интернет-адрес (URL) файла с программой:

```
<script type="text/javascript" src="script.js"></script>
```

- defer — программа, расположенная во внешнем файле, будет выполнена только после полной загрузки Web-страницы:

```
<script src="script.js" defer></script>
```

- async — программа, расположенная во внешнем файле, может быть выполнена асинхронно. По умолчанию Web-браузер выполняет инструкции построчно и, дойдя до тега <script>, загружает и разбирает программу, не выполняя при этом разбор HTML-кода. Иными словами, пока программа не загрузится в окне Web-браузера, пользователь ничего не увидит. Параметр async снимает эту блокировку и выполняет обе операции параллельно. Параметр доступен только в HTML 5:

```
<script src="script.js" async></script>
```

Если Web-браузер не поддерживает JavaScript, или выполнение скриптов запрещено в настройках Web-браузера, то будет выведен текст между тегами <noscript> и </noscript>.

Ранее содержимое тега <script> заключалось также в теги HTML-комментария <!-- и -->:

```
<script type="text/javascript">
<!--
document.write("Hello, world");
//-->
</script>
```

поскольку Web-браузеры, не поддерживающие JavaScript, выводят код скрипта в виде обычного текста.

Дело в том, что интерпретатор JavaScript игнорирует открывающий тег HTML-комментария (<!--), т. к. никакая строка программы JavaScript не может начинаться с символа <. Но закрывающий тег HTML-комментария (-->), начинающийся с двух минусов (--), распознается интерпретатором как ошибка, т. к. в JavaScript имеется предопределенный оператор --. По этой причине перед закрывающим тегом необходимо поставить символы комментария языка JavaScript (//):

```
//-->
```

> **Примечание**
>
> В настоящее время практически все Web-браузеры распознают тег <script>. Поэтому особого смысла заключать программу в символы комментария нет.

3.1.3. Местоположение программы

В листинге 3.1 мы поместили тег `<script>` внутри раздела `BODY` и для вывода надписи «Hello, world» в окно Web-браузера воспользовались методом `write()` встроенного объекта `document`:

```
document.write("Hello, world");
```

Доступ к методу или свойству объекта осуществляется с помощью точечной нотации. Параметры, передаваемые методу, указываются внутри круглых скобок, расположенных после названия метода. В нашем примере мы передаем методу `write()` строку `"Hello, world"`. Текст строки указывается внутри кавычек. Обратите внимание: каждая инструкция в JavaScript заканчивается точкой с запятой. Это можно сравнить с точкой в конце предложения.

> ***Примечание***
>
> Необходимо заметить, что это необязательное требование. Если инструкция расположена на отдельной строке, то точку с запятой можно не указывать. Тем не менее рекомендуется указывать точку с запятой в конце каждой инструкции. Это позволит избежать множества ошибок в дальнейшем.

Тег `<script>` наиболее часто располагается в разделе `HEAD` (листинг 3.2).

Листинг 3.2. Размещение программы в разделе `HEAD`

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Размещение программы в разделе HEAD</title>
   <script>
      function printHelloWorld() {
         var div1 = document.getElementById("div1");
         div1.innerHTML = "Hello, world";
      }
   </script>
</head>
<body onload="printHelloWorld()">
   <div id="div1">Ваш Web-браузер не поддерживает JavaScript</div>
</body>
</html>
```

В этом случае пользоваться методом `write()` нельзя, т. к. он используется только при формировании содержимого документа во время загрузки страницы. Чтобы иметь возможность вывести что-либо в окно Web-браузера, вначале нужно получить ссылку на какой-либо элемент с помощью метода `getElementById()` объекта `document`, а затем воспользоваться свойством `innerHTML`:

```
var div1 = document.getElementById("div1");
div1.innerHTML = "Hello, world";
```

Для создания уникального идентификатора элемента нужно воспользоваться параметром `id`, который имеют все теги:

```
<div id="div1"></div>
```

Следует также учитывать, что по умолчанию код программы, расположенный в разделе `HEAD`, выполняется раньше, чем сформируется структура HTML-документа. В итоге мы будем получать ссылку на элемент, которого еще не существует. Чтобы этого избежать, код программы размещен внутри тела функции `printHelloWorld()` (внутри фигурных скобок). В этом случае функция будет загружена, но код внутри нее будет выполнен только при вызове функции. Когда же функция будет вызвана? Получить доступ к элементу страницы можно только после полной загрузки страницы, т. е. при наступлении события окончания загрузки страницы. Задать обработчик этого события позволяет параметр `onload`, размещенный в теге `<body>`:

```
<body onload="printHelloWorld()">
```

Если программа расположена внутри раздела `BODY` после кода создания элемента, например в самом конце раздела `BODY`, то назначать обработчик события окончания загрузки страницы не нужно, т. к. элемент уже существует, и мы можем к нему обратиться:

```
<div id="div1">Ваш Web-браузер не поддерживает JavaScript</div>
<script>
   var div1 = document.getElementById("div1");
   div1.innerHTML = "Hello, world";
</script>
```

Программа может быть расположена в отдельном файле с расширением js (листинг 3.3). В этом случае файл подключается с помощью параметра `src` тега `<script>` (листинг 3.4).

Листинг 3.3. Содержимое файла script.js

```
function printHelloWorld() {
   var div1 = document.getElementById("div1");
   div1.innerHTML = "Hello, world";
}
```

Листинг 3.4. Размещение программы в отдельном файле

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Размещение программы в отдельном файле</title>
   <script src="script.js"></script>
</head>
```

```
<body onload="printHelloWorld()">
   <div id="div1">Ваш Web-браузер не поддерживает JavaScript</div>
</body>
</html>
```

Здесь мы также заключили код внутри тела функции и указали ее в качестве обработчика события `onload`.

Если для тега `<script>` указан параметр `defer` (листинг 3.6), то код внутри файла script.js (листинг 3.5) будет выполнен только после полной загрузки страницы. В этом случае назначать обработчик для события `onload` не нужно.

Листинг 3.5. Содержимое файла script.js

```
var div1 = document.getElementById("div1");
div1.innerHTML = "Hello, world";
```

Листинг 3.6. Параметр `defer`

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Параметр defer</title>
   <script src="script.js" defer></script>
</head>
<body>
   <div id="div1">Ваш Web-браузер не поддерживает JavaScript</div>
</body>
</html>
```

Опять-таки, если код загрузки программы расположен после кода создания элемента, к которому мы обращаемся, и не расположен внутри тела функции, то обработчик события `onload` не нужен:

```
<div id="div1">Ваш Web-браузер не поддерживает JavaScript</div>
<script src="script.js"></script>
```

3.1.4. Консоль в Web-браузере Firefox

Как вы уже знаете, Web-браузер Firefox содержит **Инструменты разработчика**. В состав этого средства входит **Веб-консоль**, в которую выводятся различные сообщения — например, при наличии ошибок в программе. Чтобы открыть консоль, переходим в главное меню и выбираем пункт **Разработка | Веб-консоль** или нажимаем комбинацию клавиш <Ctrl>+<Shift>+<K>. В результате отобразится содержимое вкладки **Консоль**.

В верхней части вкладки **Консоль** расположены переключатели **Сеть**, **CSS**, **JS**, **Защита**, **Журнал** и **Сервер**, с помощью которых можно включить или отключить вывод сообщений в окно консоли. Убедитесь, что все переключатели подсвечены светло-синем цветом. Если фон переключателя белого цвета, то вывод сообщений отключен. Держите консоль всегда открытой при написании и отладке программы, и вы сразу заметите наличие в программе ошибок.

Консоль можно также использовать для вывода отладочной информации. К этому способу мы будем очень часто прибегать для вывода результатов выполнения учебных скриптов. Для вывода сообщения в консоль служит такой код:

```
<script>
   console.log("Hello, world");
</script>
```

Сообщение **Hello, world** отобразится на вкладке **Консоль**, если переключатель **Журнал** подсвечен светло-синем цветом.

Нас будет также интересовать вкладка **Отладчик**, которая открывается при наличии ошибок в процессе загрузки страницы. На этой вкладке существует возможность выполнять программу по шагам, отслеживая значения различных переменных, — очень удобное средство для поиска ошибок в программе, особенно логических.

Web-браузер Firefox содержит также **Простой редактор JavaScript**, который можно использовать при выполнении скриптов в контексте текущей Web-страницы, получая и изменяя свойства элементов. Чтобы открыть редактор, переходим в главное меню и выбираем пункт **Разработка | Простой редактор JavaScript** или нажимаем комбинацию клавиш <Shift>+<F4>. Редактор подсвечивает код разными цветами, а также позволяет закончить слово или вывести список возможных вариантов. Для этого нужно ввести первые буквы слова и нажать комбинацию клавиш <Ctrl>+<Пробел>. Чтобы получить список свойств и методов объекта, надо после ввода точки нажать комбинацию клавиш <Ctrl>+<Пробел>. Для запуска программы нажимаем кнопку **Запустить** или комбинацию клавиш <Ctrl>+<R>.

3.1.5. Комментарии в JavaScript

Все, что расположено после символов `//` до конца строки, в JavaScript считается *однострочным комментарием*:

```
// Однострочный комментарий
```

Однострочный комментарий можно записать после инструкции:

```
document.write("Hello, world"); // Однострочный комментарий
```

Кроме того, существует *многострочный комментарий*. Он начинается с символов `/*` и заканчивается символами `*/`:

```
/*
Многострочный комментарий
*/
```

3.1.6. Окно с сообщением и кнопкой *OK*

Прежде чем начинать изучение языка JavaScript, рассмотрим встроенные диалоговые окна, которые позволят нам выводить результаты работы программы или значения переменных, а также вводить данные.

Метод `alert()` отображает диалоговое окно с сообщением и кнопкой **OK**. В листинге 3.7 демонстрируется вывод приветствия с помощью метода `alert()`.

Листинг 3.7. Метод `alert()`

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Метод alert()</title>
</head>
<body>
<script>
   window.alert("Hello, world");
</script>
</body>
</html>
```

Объект `window` перед методом `alert()` указывать необязательно:

```
alert("Hello, world");
```

Сообщение можно разбить на строки с помощью последовательности символов `\n`:

```
alert("Строка1\nСтрока2\n\nСтрока4");
```

3.1.7. Окно с сообщением и кнопками *OK* и *Cancel*

Метод `confirm()` отображает диалоговое окно с сообщением и двумя кнопками **OK** и **Cancel** (листинг 3.8). Он возвращает логическое значение `true`, если нажата кнопка **OK**, и `false` — если **Cancel**.

Листинг 3.8. Метод `confirm()`

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Метод confirm()</title>
</head>
<body>
<script>
```

```
if (window.confirm("Нажмите одну из кнопок")) {
   alert("Нажата кнопка OK");
}
else {
   alert("Нажата кнопка Cancel");
}
</script>
</body>
</html>
```

Объект `window` перед методом `confirm()` указывать необязательно:

```
if (confirm("Нажмите одну из кнопок")) { ... }
```

3.1.8. Окно с полем ввода и кнопками *OK* и *Cancel*

Метод `prompt()` отображает диалоговое окно с сообщением, полем ввода и двумя кнопками **OK** и **Cancel** (листинг 3.9). Он возвращает введенное значение, если нажата кнопка **OK**, или специальное значение `null`, если нажата кнопка **Cancel**.

Листинг 3.9. Метод `prompt()`

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Метод prompt()</title>
</head>
<body>
<script>
var n = window.prompt("Введите ваше имя", "Это значение по умолчанию");
if (n === null) {
   document.write("Вы нажали Cancel");
}
else {
   document.write("Привет " + n);
}
</script>
</body>
</html>
```

Объект `window` перед методом `prompt()` указывать необязательно:

```
var n = prompt("Введите ваше имя", "Это значение по умолчанию");
```

3.1.9. JavaScript-библиотеки

Мы уже не раз упоминали, что разные Web-браузеры могут по-разному выполнять код программы. По этой причине при написании приложений приходится учитывать особенности каждого Web-браузера. Проблема заключается в том, что установить все версии каждого Web-браузера на один компьютер практически невозможно, а значит, обеспечить полную кроссбраузерность самостоятельно не получится.

При использовании библиотек любой программист может сообщить о проблеме в каком-либо Web-браузере, а разработчик библиотеки, опираясь на это сообщение, имеет возможность обработать ошибку. После исправления ошибки всем остальным программистам достаточно сменить версию библиотеки. Таким образом, используя возможности какой-либо библиотеки, можно забыть о проблеме с кроссбраузерностью приложения.

Наиболее часто используются следующие JavaScript-библиотеки:

- jQuery — **http://jquery.com/**;
- Prototype — **http://prototypejs.org/**;
- MooTools — **https://mootools.net/**;
- Dojo — **http://dojotoolkit.org/**;
- Yahoo! UI Library (YUI) — **https://yuilibrary.com/**.

Из этого списка хотим особо выделить библиотеку jQuery, предоставляющую функциональность, которую может использовать практически любой разработчик. Она обеспечивает кроссбраузерную поддержку приложений, имеет небольшой размер и не засоряет глобальное пространство имен тривиальными идентификаторами. Большой популярности jQuery способствовали также дополнительные модули, реализующие готовые компоненты или добавляющие новую функциональность. Например, библиотека jQuery UI добавляет возможность перемещения и изменения размеров любых элементов с помощью мыши, позволяет сортировать и выделять элементы, а также предоставляет готовые компоненты ("аккордеон", панель с вкладками, диалоговые окна, календарь и др.).

Давайте попробуем подключить библиотеку jQuery и вывести приветствие с ее помощью (листинг 3.10). Предварительно нужно скачать библиотеку со страницы **http://jquery.com/download/**. Скачиваем файл, например, с названием jquery-3.2.1.min.js, размещаем его в одной папке с HTML-документом, а затем переименовываем в jquery.js.

Листинг 3.10. Вывод приветствия с помощью библиотеки jQuery

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Вывод приветствия с помощью библиотеки jQuery</title>
   <script src="jquery.js"></script>
```

```
   <script>
   $(function() {
      $("#div1").text("Hello, world");
   });
   </script>
</head>
<body>
   <div id="div1"></div>
</body>
</html>
```

Этот код аналогичен коду из листинга 3.2, только он гораздо проще. Вначале, в разделе `HEAD`, мы подключаем библиотеку jQuery. Затем создаем анонимную функцию, которая будет вызвана, когда структура HTML-документа будет сформирована. Причем код функции будет выполнен, не дожидаясь загрузки других элементов Web-страницы, например изображений.

Всю функциональность в библиотеке jQuery выполняет функция `$()`. Она имеет псевдоним `jQuery()`:

```
jQuery(function() {
   jQuery("#div1").text("Hello, world");
});
```

Других идентификаторов в глобальную область видимости библиотека не добавляет, тем самым не засоряя ее.

Функция `$()` имеет несколько форматов. Наиболее часто используемый формат:

```
$(<CSS-селектор>)
```

В качестве параметра мы можем задействовать CSS-селекторы, которые мы уже изучили в предыдущей главе. В листинге 3.10 используется доступ к элементу по идентификатору:

```
$("#div1").text("Hello, world");
```

Функция возвращает ссылку на один элемент или на целый набор элементов. Используя различные методы, можно выполнять над элементами те или иные операции. Если функция возвращает набор элементов, то операции в большинстве случаев выполняются над всеми элементами из набора. Например, выведем текст приветствия во всех абзацах:

```
<p></p>
<p></p>
<p></p>
<script>
$(function() {
   $("p").text("Hello, world");
});
</script>
```

Чтобы вывести текст только в первом абзаце, достаточно дополнительно указать псевдокласс `:first`:

```
$("p:first").text("Hello, world");
```

Пользоваться библиотекой jQuery очень просто и очень эффективно, а главное — не нужно думать, будет ли код работать в каком-либо Web-браузере.

3.2. Переменные и типы данных

Переменные — это участки памяти, используемые программой для хранения данных.

3.2.1. Именование переменных

Каждая переменная должна иметь в программе уникальное имя, состоящее из латинских букв, цифр и знаков подчеркивания. Первым символом может быть либо буква, либо знак подчеркивания. В названии переменной может также присутствовать символ `$`. Имена переменных не должны совпадать с зарезервированными ключевыми словами языка JavaScript.

Правильные имена переменных:

```
x, strName, y1, _name, window1
```

Неправильные имена переменных:

```
1y, ИмяПеременной, window
```

Последнее имя неправильное, потому что является ключевым словом.

При указании имени переменной важно учитывать регистр букв: `strName` и `strname` — разные переменные.

3.2.2. Объявление переменной

В программе переменные объявляются с помощью ключевого слова `var`:

```
var x;
```

Можно объявить сразу несколько переменных в одной строке, указав их через запятую:

```
var x, y, z;
```

3.2.3. Типы данных и инициализация переменных

В JavaScript переменные могут содержать следующие типы данных:

- `number` — целые числа или числа с плавающей точкой (дробные числа):

  ```
  console.log(typeof 1);              // number
  console.log(typeof 2.0);            // number
  ```

- `string` — строки:

  ```
  console.log(typeof 'строка');       // string
  ```

- `boolean` — логический тип данных. Может содержать значения `true` (истина) или `false` (ложь):

  ```
  console.log(typeof true);           // boolean
  ```

- `function` — функции. В языке JavaScript ссылку на функцию можно присвоить какой-либо переменной. Для этого название функции указывается без круглых скобок. Кроме того, функции имеют свойства и методы:

  ```
  console.log(typeof function() {}); // function
  ```

- `object` — массивы, объекты, а также переменная со значением `null`:

  ```
  console.log(typeof [1, 2, 3]);      // object
  console.log(typeof window);         // object
  console.log(typeof null);           // object
  ```

- `undefined` — неопределенное значение:

  ```
  var str;
  console.log(typeof str);            // undefined
  ```

В этих примерах мы воспользовались оператором `typeof`, который возвращает строку, описывающую тип данных переменной. Все значения будут выведены в консоль Web-браузера. Не забудьте открыть консоль (см. *разд. 3.1.4*), чтобы видеть результаты выполнения учебных примеров.

При инициализации переменной JavaScript автоматически относит переменную к одному из типов данных. Что такое *инициализация переменных*? Это операция присвоения переменной начального значения.

Значение переменной присваивается с помощью оператора =:

```
var x;
x = 5;
console.log(typeof x);          // number
console.log(x);                 // 5
```

Переменной может быть присвоено начальное значение сразу при ее объявлении:

```
var y = 12.5;
console.log(typeof y);          // number
console.log(y);                 // 12.5
```

При объявлении можно задать начальные значения сразу нескольким переменным:

```
var x = 5, y = 12.5, str = "Строка";
console.log(x);                 // 5
console.log(y);                 // 12.5
console.log(str);               // Строка
```

3.2.4. Проверка существования переменной

Если в программе обратиться к переменной, которая не объявлена, то возникнет критическая ошибка. Если переменная объявлена, но ей не присвоено начальное значение, то значение предполагается равным `undefined`:

```
var x;
console.log(x);    // undefined
console.log(str);  // ReferenceError: str is not defined
```

Проверить существование глобальной переменной можно следующим способом:

```
if (window.str === undefined) {
   console.log("Переменная не существует");
}
else {
   console.log("Переменная существует");
}
```

В этом примере, чтобы избежать ошибки обращения к несуществующей переменной, мы воспользовались встроенным объектом `window`, через который доступны все глобальные переменные. Обращение через объект к несуществующему свойству ошибки не генерирует.

Пример обращения к глобальной переменной через объект `window`:

```
var x = 10;
console.log(window.x);    // 10
```

Также мы воспользовались глобальным свойством `undefined`, которое содержит значение `undefined`. Название этого свойства не является зарезервированным ключевым словом, поэтому можно определить локальную переменную с таким именем, а вот изменить значение глобального свойства нельзя.

Для проверки можно также воспользоваться оператором `typeof`:

```
if (typeof str == "undefined") {
   console.log("Переменная не существует");
}
else {
   console.log("Переменная существует");
}
```

В операторе `typeof` можно просто указать название переменной без объекта `window`, т. к. ошибка обращения к несуществующей переменной в этом случае не генерируется.

3.2.5. Константы

Константы служат для хранения значений, которые не должны изменяться во время работы программы. Создать константу можно с помощью ключевого слова `const`:

```
const <Имя константы> = <Значение константы>;
```

Значение можно присвоить константе только при объявлении. После объявления изменить значение константы нельзя:

```
const MY_CONST = 10;
console.log( MY_CONST );          // 10
```

Обратите внимание: в названии константы принято использовать буквы только в верхнем регистре. Если название константы состоит из нескольких слов, то между словами указывается символ подчеркивания. Это позволяет отличить внутри программы константу от обычной переменной.

Как и при объявлении переменной, можно в одной инструкции объявить несколько констант через запятую:

```
const MY_CONST1 = 10, MY_CONST2 = 5, MY_CONST3 = 88;
```

Если мы присваиваем константе объект, то заменить этот объект другим нельзя, а вот изменить свойства объекта можно:

```
const MY_CONST = [1, 2, 3];
console.log( MY_CONST );          // Array [ 1, 2, 3 ]
MY_CONST[0] = 88;
console.log( MY_CONST );          // Array [ 88, 2, 3 ]
```

Полная поддержка ключевого слова `const` появилась в браузерах Internet Explorer 11, Firefox 36.0, Chrome 49.0, Opera 36.0 и Safari 10.0.

3.3. Операторы JavaScript

Операторы позволяют выполнить с данными определенные действия. Например, операторы присваивания служат для сохранения данных в переменной, математические операторы позволяют произвести арифметические вычисления, а оператор конкатенации строк используется для соединения двух строк в одну. Операторы берут одно или два значения, представляющие собой переменную, константу или другое выражение, содержащее операторы или функции, и возвращают одно значение, определяемое по исходным данным. Рассмотрим доступные в JavaScript операторы подробно.

3.3.1. Математические операторы

В JavaScript доступны следующие математические операторы:

- `+` — сложение:

  ```
  console.log( 10 + 15 );          // 25
  ```

- `-` — вычитание:

  ```
  console.log( 30 - 15 );          // 15
  ```

- `*` — умножение:

  ```
  console.log( 12 * 10 );          // 120
  ```

- `/` — деление:

  ```
  console.log( 10 / 2 );             // 5
  console.log( 10 / 3 );             // 3.3333333333333335
  ```

  Деление числа на `0` приведет к значению плюс или минус `Infinity` (бесконечность), а деление числа `0` на `0` — к значению `NaN` (нет числа):

  ```
  console.log(  10 / 0 );            // Infinity
  console.log( -10 / 0 );            // -Infinity
  console.log(   0 / 0 );            // NaN
  ```

- `%` — остаток от деления:

  ```
  console.log( 10 % 2 );             // 0
  console.log( 10 % 3 );             // 1
  console.log( 10 % 4 );             // 2
  ```

- `**` — возведение в степень (оператор доступен только в последних версиях Web-браузеров, в старых версиях нужно использовать метод `pow()`):

  ```
  console.log( 10 ** 2 );            // 100
  console.log( 3 ** 3 );             // 27
  console.log( -(3 ** 3) );          // -27
  console.log(Math.pow(10, 2));      // 100
  ```

- `-` — унарный минус:

  ```
  var x = 10;
  console.log( -x );            // -10
  ```

- `+` — унарный плюс (преобразует значение в тип `number`):

  ```
  var n = "10";
  var x = +n;
  console.log( x );            // 10
  console.log( typeof x );     // number
  ```

- `++` — оператор инкремента (увеличивает значение переменной на `1`):

  ```
  var x = 10;
  x++;                         // Эквивалентно x = x + 1;
  console.log( x );            // 11
  ```

- `--` — оператор декремента (уменьшает значение переменной на `1`):

  ```
  var x = 10;
  x--;                         // Эквивалентно x = x - 1;
  console.log( x );            // 9
  ```

Операторы инкремента и декремента могут использоваться в постфиксной или префиксной формах:

```
x++; x--; // Постфиксная форма
++x; --x; // Префиксная форма
```

В чем разница? При постфиксной форме (x++) возвращается значение, которое переменная имела перед операцией, а при префиксной форме (++x) — вначале выполняется операция и только потом возвращается значение. Продемонстрируем разницу на примере (листинг 3.11).

Листинг 3.11. Постфиксная и префиксная формы

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Постфиксная и префиксная формы</title>
</head>
<body>
<script>
   var x = 5, y;
   y = x++;         // y = 5, x = 6
   var msg;
   msg = "<b>Постфиксная форма (y = x++;):<" + "/b><br> y = ";
   msg += y + "<br>x = " + x + "<br><br>";
   x = 5;
   y = ++x;         // y = 6, x = 6
   msg += "<b>Префиксная форма (y = ++x;):<" + "/b><br> y = ";
   msg += y + "<br>x = " + x;
   document.write(msg);
</script>
</body>
</html>
```

3.3.2. Побитовые операторы

В JavaScript доступны следующие побитовые операторы:

- ~ — двоичная инверсия. Значение каждого бита заменяется на противоположное:

```
y = ~x;
```

- & — двоичное И:

```
var x = 100, y = 75, z;
console.log( x.toString(2) );           // 1100100
console.log( y.toString(2) );           // 1001011
z = x & y;
console.log( z.toString(2) );           // 1000000
```

- | — двоичное ИЛИ:

```
var x = 100, y = 75, z;
console.log( x.toString(2) );           // 1100100
```

```
console.log( y.toString(2) );            // 1001011
z = x | y;
console.log( z.toString(2) );            // 1101111
```

- ^ — двоичное исключающее ИЛИ:

```
var x = 100, y = 250, z;
console.log( x.toString(2) );            // 01100100
console.log( y.toString(2) );            // 11111010
z = x ^ y;
console.log( z.toString(2) );            // 10011110
```

- << — сдвиг влево (сдвиг влево на один или более разрядов с заполнением младших разрядов нулями):

```
var x = 100;
console.log( x.toString(2) );            // 001100100
x = x << 1;
console.log( x.toString(2) );            // 011001000
x = x << 1;
console.log( x.toString(2) );            // 110010000
```

- >> — сдвиг вправо (сдвиг вправо на один или более разрядов с заполнением старших разрядов содержимым самого старшего разряда):

```
x = x >> 1;
```

- >>> — сдвиг вправо без учета знака (сдвиг вправо на один или более разрядов с заполнением старших разрядов нулями):

```
x = x >>> 1;
```

Как следует из названия, побитовые операторы выполняют поразрядные действия с двоичным представлением целых чисел.

3.3.3. Операторы присваивания

Операторы присваивания предназначены для сохранения значения в переменной. В JavaScript доступны следующие операторы присваивания:

- = — присваивает переменной значение:

```
x = 5;
```

Оператор = можно также использовать для распаковки объектов:

```
var a = 0, b = 0, arr = [1, 2];
[a, b] = arr;
console.log( a + " " + b );   // 1 2
```

- += — увеличивает значение переменной на указанную величину:

```
x += 5; // Эквивалентно x = x + 5;
```

- -= — уменьшает значение переменной на указанную величину:

```
x -= 5; // Эквивалентно x = x - 5;
```

- `*=` — умножает значение переменной на указанную величину:

```
x *= 5; // Эквивалентно x = x * 5;
```

- `/=` — делит значение переменной на указанную величину:

```
x /= 5; // Эквивалентно x = x / 5;
```

- `%=` — делит значение переменной на указанную величину и возвращает остаток:

```
x %= 5; // Эквивалентно x = x % 5;
```

- `&=`, `|=`, `^=`, `<<=`, `>>=` и `>>>=` — побитовые операторы с присваиванием.

3.3.4. Операторы сравнения

Операторы сравнения используются в логических выражениях. В JavaScript доступны следующие операторы сравнения:

- `==` — равно;
- `===` — строго равно;
- `!=` — не равно;
- `!==` — строго не равно;
- `<` — меньше;
- `>` — больше;
- `<=` — меньше или равно;
- `>=` — больше или равно.

Логические выражения возвращают только два значения: `true` (истина) или `false` (ложь):

```
console.log( 10 == 10 );    // true
console.log( 10 == 5  );    // false
console.log( 10 != 5  );    // true
console.log( 10 <  5  );    // false
console.log( 10 >  5  );    // true
console.log( 10 <= 5  );    // false
console.log( 10 >= 5  );    // true
```

В чем отличие оператора `==` (равно) от оператора `===` (строго равно)? Дело все в том, что если используется оператор `==`, интерпретатор пытается преобразовать разные типы данных к одному и лишь затем сравнивает их. Оператор `===`, встретив данные разных типов, сразу возвращает `false` (ложь):

```
console.log( 5 == '5'  );  // true
console.log( 5 === '5' );  // false
```

Кроме того, значение логического выражения можно инвертировать с помощью оператора `!` таким образом:

```
console.log( 10 == 10 );    // true
console.log( !(10 == 10) ); // false
```

Если значения равны, то возвращается значение `true`, но так как перед выражением стоит оператор `!`, выражение вернет `false`.

Несколько логических выражений можно объединить в одно большое с помощью следующих операторов:

- `&&` — логическое `И`. Логическое выражение вернет `true` только в случае, если оба подвыражения вернут `true`:

```
console.log( (10 == 10) && (5 != 3) ); // true
console.log( (10 == 10) && (5 == 3) ); // false
```

- `||` — логическое `ИЛИ`. Логическое выражение вернет `true`, если хотя бы одно из подвыражений вернет `true`. Если первое подвыражение вернет значение `true`, то второе подвыражение даже не будет вычисляться:

```
console.log( (10 == 10) || (5 != 3) ); // true
console.log( (10 == 10) || (5 == 3) ); // true
```

Оператор `||` также часто используется для создания необязательных параметров в функции. Если первое выражение не может быть преобразовано в `true`, то возвращается значение второго выражения:

```
function test(str) {
   str = str || "Значение по умолчанию";
   return str;
}
console.log( test() );                     // Значение по умолчанию
console.log( test("Значение указано") ); // Значение указано
```

3.3.5. Приоритет выполнения операторов

В какой последовательности будет вычисляться следующее выражение?

```
var x = 5 + 10 * 3 / 2;
console.log( x ); // 20
```

Это зависит от приоритета выполнения операторов. В данном случае последовательность вычисления выражения будет следующей:

1. Число `10` будет умножено на `3`, т. к. приоритет оператора умножения выше приоритета оператора сложения.
2. Полученное значение будет поделено на `2`, поскольку приоритет оператора деления равен приоритету оператора умножения (а операторы с равными приоритетами выполняются слева направо), но выше, чем у оператора сложения.
3. К полученному значению будет прибавлено число `5`, т. к. оператор присваивания `=` имеет наименьший приоритет.
4. Значение будет присвоено переменной `x`.

Последовательность вычисления выражения можно изменить с помощью скобок:

```
var x = (5 + 10) * 3 / 2;
console.log( x ); // 22.5
```

Теперь порядок вычислений будет другим:

1. К числу 5 будет прибавлено 10.
2. Полученное значение будет умножено на 3.
3. Полученное значение будет поделено на 2.
4. Значение будет присвоено переменной x.

Приведем операторы в порядке убывания приоритета:

- ++, -- — постфиксный инкремент, постфиксный декремент;
- !, ~, ++, --, +, - — отрицание, двоичная инверсия, префиксный инкремент, префиксный декремент, унарный плюс, унарный минус;
- ** — возведение в степень;
- *, /, % — умножение, деление, остаток от деления;
- +, - — сложение и вычитание;
- <<, >>, >>> — двоичные сдвиги;
- <, <=, >, >= — операторы сравнения;
- ==, !=, ===, !== — операторы сравнения;
- & — двоичное И;
- ^ — двоичное исключающее ИЛИ;
- | — двоичное ИЛИ;
- =, +=, -=, *=, /=, %= — присваивание;
- , — оператор «запятая».

3.3.6. Преобразование типов данных

Что будет, если к числу прибавить строку?

```
var str = "5", x = 3;
var var1 = x + str;       // Переменная содержит строку "35"
console.log(var1);        // 35
console.log(typeof var1); // string
var var2 = str + x;       // Переменная содержит строку "53"
console.log(var2);        // 53
console.log(typeof var2); // string
```

В этом случае интерпретатор столкнется с несовместимостью типов данных и попытается преобразовать переменные к одному типу данных, а затем выполнить операцию. В нашем случае переменная x, имеющая тип number (число), будет преобразована к типу string (строка), а затем будет произведена операция конкатенации строк.

А что будет, если из числа вычесть строку, число умножить на строку или число разделить на строку?

```
var str = "5", x = 15;
var var1 = x - str;       // Переменная содержит число 10
console.log(var1);        // 10
console.log(typeof var1); // number
var var2 = x * str;       // Переменная содержит число 75
console.log(var2);        // 75
console.log(typeof var2); // number
var var3 = x / str;       // Переменная содержит число 3
console.log(var3);        // 3
console.log(typeof var3); // number
```

Интерпретатор попытается преобразовать строку в число, а затем вычислить выражение. Причем не важно, в какой последовательности будут указаны число и строка:

```
var var4 = str * x;       // Переменная содержит число 75
console.log(var4);        // 75
console.log(typeof var4); // number
```

Но что будет, если в строке будут одни буквы?

```
var str = "Строка", x = 15;
var var1 = x - str;       // Переменная содержит значение NaN
console.log(var1);        // NaN
console.log(typeof var1); // number
```

В этом случае интерпретатор не сможет преобразовать строку в число и присвоит переменной значение `NaN` (*Not a Number*, не число).

С одной стороны, хорошо, что интерпретатор делает преобразование типов данных за нас. Но с другой стороны, можно получить результат, который вовсе не планировался. По этой причине лучше оперировать переменными одного типа, а если необходимо выполнить преобразования типов, то делать это самим.

Для преобразования типов данных можно использовать следующие встроенные функции JavaScript:

- `parseInt(<Строка>[, <Основание>])` — преобразует строку в целое число. Строка считается заданной в системе счисления, указанной вторым необязательным параметром. Если основание не указано, то система счисления выбирается автоматически и зависит от префиксов в строке, — например, если строка начинается с "`0x`", то будет использоваться основание `16`. Если строка не может быть преобразована в число, возвращается значение `NaN`:

```
console.log( parseInt("10") );          // 10
console.log( parseInt("0xFF") );        // 255
console.log( parseInt("0xFF", 16) );    // 255
console.log( parseInt("0167", 8) );     // 119
console.log( parseInt("1110111", 2) );  // 119
console.log( parseInt("строка", 2) );   // NaN
```

- `parseFloat(<Строка>)` — преобразует строку в число с плавающей точкой:

```
console.log( parseFloat("5.2") );       // 5.2
console.log( parseFloat("5e-2") );      // 0.05
console.log( parseFloat("строка") );    // NaN
```

Для преобразования числа в строку используется метод toString([<Основание>]):

```
var x = 10, y = 5.2;
console.log( x.toString() );     // 10
console.log( x.toString(8) );    // 12
console.log( y.toString() );     // 5.2
```

В качестве примера использования преобразования типов данных просуммируем два числа, введенные пользователем в поля двух диалоговых окон (листинг 3.12).

Листинг 3.12. Вычисление суммы двух чисел

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Вычисление суммы двух чисел</title>
</head>
<body>
<script>
var var1, var2, sum1, sum2, msg;
var1 = window.prompt("Введите число 1", "");
if (var1 == null) {
   document.write("Вы нажали Отмена");
}
else {
   var2 = window.prompt("Введите число 2", "");
   if (var2 == null) {
      document.write("Вы нажали Отмена");
   }
   else {
      sum1 = var1 + var2;
      msg = "До преобразования типов:<br>Значение суммы чисел ";
      msg += var1 + " и " + var2 + " равно ";
      msg += sum1 + "<br><br>";
      sum2 = parseInt(var1, 10) + parseInt(var2, 10);
      msg += "После преобразования типов:<br>";
      msg += "Значение суммы чисел " + var1 + " и ";
      msg += var2 + " равно " + sum2;
      document.write(msg);
   }
}
</script>
</body>
</html>
```

Если в обоих диалоговых окнах набрать число 5, то в окне Web-браузера отобразится следующий текст:

```
До преобразования типов:
Значение суммы чисел 5 и 5 равно 55

После преобразования типов:
Значение суммы чисел 5 и 5 равно 10
```

Итак, диалоговые окна возвращают в качестве типа значения строку. Чтобы получить сумму двух чисел, указанных в полях диалоговых окон, нужно обязательно выполнить преобразование типов, иначе мы получим еще одну строку, а не сумму.

3.3.7. Оператор ветвления *if...else*

Условные операторы позволяют в зависимости от значения логического выражения выполнить отдельный участок программы или, наоборот, не выполнять его. Логические выражения возвращают только два значения: `true` (истина) или `false` (ложь).

Оператор ветвления `if...else` имеет следующий формат:

```
if (<Логическое выражение>) {
   <Блок, выполняемый, если условие истинно>
}
[else {
   <Блок, выполняемый, если условие ложно>
}]
```

Оператор ветвления `if...else` мы уже использовали ранее в наших примерах — например, чтобы проверить, какая из кнопок диалогового окна нажата (см. листинг 3.8). Так как при нажатии кнопки **OK** возвращается значение `true`, то можно узнать, какая кнопка нажата, используя оператор ветвления `if...else`:

```
if (window.confirm("Нажмите одну из кнопок")) {
   alert("Нажата кнопка OK");
}
else {
   alert("Нажата кнопка Cancel");
}
```

Обратите внимание: логическое выражение не содержит операторов сравнения:

```
if (window.confirm("Нажмите одну из кнопок")) {
```

Такая запись эквивалентна записи:

```
if (window.confirm("Нажмите одну из кнопок") == true) {
```

Проверка на равенство выражения значению `true` (истина) выполняется по умолчанию.

В качестве логического выражения можно указать просто значение, которое будет преобразовано в `false` в следующих случаях:

- если число равно `0` или `0.0`:

  ```
  console.log( !!0 );         // false
  console.log( !!0.0 );       // false
  ```

- если указана пустая строка:

  ```
  console.log( !!"" );        // false
  ```

- если переменная содержит значение `null`:

  ```
  console.log( !!null );      // false
  ```

- если переменная содержит значение `NaN`:

  ```
  console.log( !!NaN );       // false
  ```

- если свойство объекта не существует, или переменной не присвоено начальное значение:

  ```
  var obj = {}, x;
  console.log( !!obj.attr );  // false
  console.log( !!x );         // false
  console.log( !!undefined ); // false
  ```

Все остальные значения будут преобразованы в `true`:

```
console.log( !!10 );    // true
console.log( !!"0" );   // true
var obj = { n: 20 };
console.log( !!obj.n ); // true
```

Для примера напишем программу (листинг 3.13), которая проверяет, является ли введенное пользователем число четным или нет. После проверки выводится соответствующее сообщение.

Листинг 3.13. Проверка числа на четность

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Проверка числа на четность</title>
</head>
<body>
<script>
var x = window.prompt("Введите число", "");
if (x === null) {
   console.log("Вы нажали Отмена");
}
else {
   if ( (parseInt(x, 10) % 2) == 0 ) {
      console.log("Четное число");
   }
   else {
      console.log("Нечетное число");
   }
}
```

```
</script>
</body>
</html>
```

Как видно из примера, один условный оператор можно вложить в другой. Кроме того, если блок состоит из одной инструкции, фигурные скобки можно не указывать:

```
if ( (parseInt(x, 10) % 2) == 0 ) console.log("Четное число");
else console.log("Нечетное число");
```

Более того, блока `else` может не быть совсем:

```
if ( (parseInt(x, 10) % 2) == 0 ) console.log("Четное число");
```

Для проверки нескольких условий часто используется следующий синтаксис:

```
var x = 2;
if (x == 1) {
   console.log( "x = 1" );
}
else if (x == 2) {
   console.log( "x = 2" );
}
else if (x == 3) {
   console.log( "x = 3" );
}
else console.log( "Другое значение" );
```

3.3.8. Оператор *?:*

Оператор `?:` имеет следующий формат:

```
<Переменная> = (<Логическое выражение>) ? <Выражение если Истина> :
                                          <Выражение если Ложь>;
```

Перепишем нашу программу (см. листинг 3.13) и используем оператор `?:` вместо `if...else` (листинг 3.14).

Листинг 3.14. Проверка числа на четность с помощью оператора `?:`

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Проверка числа на четность</title>
</head>
<body>
<script>
var x = window.prompt("Введите число", "");
```

```
if (x === null) {
   console.log("Вы нажали Отмена");
}
else {
   var msg = ((parseInt(x, 10) % 2) == 0) ? "Четное число" :
                                              "Нечетное число";
   console.log(msg);
}
</script>
</body>
</html>
```

3.3.9. Оператор выбора *switch*

Оператор выбора `switch` имеет следующий формат:

```
switch (<Переменная или выражение>) {
   case <Значение 1>:
      <Выражение 1>;
      break;
[  case <Значение 2>:
      <Выражение 2>;
      break;
...]
[  default:
      <Выражение>;]
}
```

Перепишем нашу программу и используем оператор `switch` вместо `if...else` и `?:` (листинг 3.15).

Листинг 3.15. Проверка числа на четность с помощью оператора `switch`

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Проверка числа на четность</title>
</head>
<body>
<script>
var x = window.prompt("Введите число", "");
if (x === null) {
   console.log("Вы нажали Отмена");
}
else {
   switch ( parseInt(x, 10) % 2 ) {
```

```
   case 0:
      console.log("Четное число");
      break;
   case 1:
      console.log("Нечетное число");
      break;
   default:
      console.log("Введенное значение не является числом");
   }
}
</script>
</body>
</html>
```

Итак, оператор `switch` позволил сделать еще одну дополнительную проверку — ведь пользователь вместо числа мог ввести строку. А в этом случае функция `parseInt()` вернет значение `NaN` (Not a Number). Любая арифметическая операция со значением `NaN` вернет в качестве значения `NaN`. В предыдущих примерах мы не выполняли эту проверку, и в случае ввода строки, которую невозможно преобразовать в число, функция возвращала фразу `"Нечетное число"`. Что, согласитесь, неверно.

Вернемся к оператору `switch`. Вместо логического выражения оператор `switch` принимает переменную или выражение. В зависимости от значения переменной (или выражения) выполняется один из блоков `case`, в котором указано это значение. Если ни одно из значений не описано в блоках `case`, то выполняется блок `default`. Сравнение производится с помощью оператора === (строго равно).

Оператор `break` позволяет досрочно выйти из оператора выбора `switch`. Зачем это нужно? Если не указать оператор `break` в конце блока `case`, то будет выполняться следующий блок `case` вне зависимости от указанного значения. Если убрать все операторы `break` из нашего примера, то в результате (при вводе четного числа) в консоли отобразятся следующие значения:

```
Четное число
Нечетное число
Введенное значение не является числом
```

Иными словами, оператор `break` следует обязательно указывать в конце каждого блока `case`.

3.4. Циклы. Многократное выполнение блока кода

Предположим, нужно вывести все числа от `1` до `100` по одному на строке. Обычным способом пришлось бы писать 100 строк кода:

```
document.write("1<br>");
document.write("2<br>");
// ...
document.write("100<br>");
```

При помощи циклов то же действие можно выполнить одной строкой кода:

```
for (var i = 1; i < 101; i++) document.write(i + "<br>");
```

Иными словами, циклы позволяют выполнить одни и те же инструкции многократно.

3.4.1. Цикл *for*

Цикл `for` используется для выполнения инструкций определенное число раз. Он имеет следующий формат:

```
for (<Начальное значение>; <Условие>; <Приращение>) {
   <Инструкции>
}
```

Здесь присутствуют следующие конструкции:

- `<Начальное значение>` — присваивает переменной-счетчику начальное значение;
- `<Условие>` — содержит логическое выражение. Пока логическое выражение возвращает значение `true`, выполняются инструкции внутри цикла. Обратите внимание: логическое выражение вычисляется на каждой итерации цикла;
- `<Приращение>` — задает изменение переменной-счетчика при каждой итерации.

Последовательность работы цикла `for`:

1. Переменной-счетчику присваивается начальное значение.
2. Проверяется условие. Если оно истинно, выполняются инструкции внутри цикла, в противном случае выполнение цикла завершается.
3. Переменная-счетчик изменяется на величину, указанную в параметре `<Приращение>`.
4. Переход к п. 2.

Цикл выполняется до тех пор, пока `<Условие>` не вернет `false`. Если это не случится, цикл будет бесконечным.

`<Приращение>` может не только увеличивать значение переменной-счетчика, но и уменьшать. Выведем все числа от `100` до `1`:

```
for (var i = 100; i > 0; i--) document.write(i + "<br>");
```

`<Приращение>` может изменять значение переменной-счетчика не только на единицу. Выведем все четные числа от `1` до `100`:

```
for (var i = 2; i < 101; i += 2) document.write(i + "<br>");
```

Следует заметить, что выражение, указанное в параметре `<Условие>`, вычисляется на каждой итерации. Рассмотрим вывод элементов массива:

```
var arr = [ 1, 2, 3 ];
for (var i = 0; i < arr.length; i++) {
   if (i == 0) {
      arr.push(4); // Добавляем новые элементы
      arr.push(5); // для доказательства
   }
```

```
   document.write(arr[i] + " ");
} // Выведет: 1 2 3 4 5
```

В этом примере мы указываем свойство `length` в параметре `<Условие>`, а внутри цикла (чтобы доказать вычисление на каждой итерации) добавляем в массив новые элементы. В итоге мы получили все элементы массива, включая новые элементы. Чтобы этого избежать, следует вычисление размера массива указать в первом параметре:

```
var arr = [1, 2, 3];
for (var i = 0, c = arr.length; i < c; i++) {
   if (i == 0) {
      arr.push(4); // Добавляем новые элементы
      arr.push(5); // для доказательства
   }
   document.write(arr[i] + " ");
} // Выведет: 1 2 3
```

Все параметры цикла `for` являются необязательными. Хотя параметры можно не указывать, точки с запятой обязательно должны присутствовать. Если все параметры не указаны, то цикл будет бесконечным. Чтобы выйти из бесконечного цикла, следует использовать оператор `break`:

```
var i = 1;                  // <Начальное значение>
for ( ; ; ) {               // Бесконечный цикл
   if (i <= 100) {          // <Условие>
      document.write(i + "<br>");
      i++;                  // <Приращение>
   }
   else {
      break;                // Выходим из цикла
   }
}
```

3.4.2. Цикл *while*

Выполнение инструкций в цикле `while` продолжается до тех пор, пока логическое выражение истинно. Он имеет следующий формат:

```
<Начальное значение>;
while (<Условие>) {
   <Инструкции>;
   <Приращение>;
}
```

Последовательность работы цикла `while`:

1. Переменной-счетчику присваивается начальное значение.
2. Проверяется условие. Если оно истинно, выполняются инструкции внутри цикла, иначе выполнение цикла завершается.

3. Переменная-счетчик изменяется на величину, указанную в параметре `<Приращение>`.
4. Переход к п. 2.

Выведем все числа от 1 до 100, используя цикл while (листинг 3.16).

Листинг 3.16. Цикл while

```
var i = 1;
while (i < 101) {
   document.write(i + "<br>");
   i++;
}
```

> **ВНИМАНИЕ!**
> Если `<Приращение>` не указано, цикл будет бесконечным.

3.4.3. Цикл *do...while*

Инструкции в теле цикла do...while выполняются до тех пор, пока логическое выражение истинно. Но, в отличие от цикла while, условие проверяется не в начале цикла, а в конце. Поэтому инструкции внутри цикла do...while выполнятся минимум один раз.

Формат цикла do...while:

```
<Начальное значение>;
do {
   <Инструкции>;
   <Приращение>;
} while (<Условие>);
```

Последовательность работы цикла do...while:

1. Переменной-счетчику присваивается начальное значение.
2. Выполняются инструкции внутри цикла.
3. Переменная-счетчик изменяется на величину, указанную в `<Приращение>`.
4. Проверяется условие. Если оно истинно, происходит переход к п. 2, а если нет — выполнение цикла завершается.

Выведем все числа от 1 до 100, используя цикл do...while (листинг 3.17).

Листинг 3.17. Цикл do...while

```
var i = 1;
do {
   document.write(i + "<br>");
   i++;
} while (i < 101);
```

ВНИМАНИЕ!

Если `<Приращение>` не указано, цикл будет бесконечным.

3.4.4. Цикл *for...in*

Цикл `for...in` используется для перебора перечислимых свойств объектов и ключей ассоциативных массивов. Он имеет следующий формат:

```
for (<Переменная> in <Объект>) {
   <Тело цикла>
}
```

Цикл `for...in` на каждой итерации присваивает параметру `<Переменная>` имя свойства объекта или ключа ассоциативного массива. С помощью ключа можно получить значение соответствующего элемента ассоциативного массива (листинг 3.18).

Листинг 3.18. Цикл `for...in`

```
var arr = new Object();
arr["Один"] = 1;
arr["Два"] = 2;
arr["Три"] = 3;
for (var key in arr) {
   // Переменной key на каждой итерации присваивается
   // ключ текущего элемента ассоциативного массива
   document.write(key + " = " + arr[key]  + "<br>");
}
```

В итоге мы получим следующий результат:

```
Один = 1
Два = 2
Три = 3
```

Следует учитывать, что порядок обхода является произвольным, поэтому перебирать обычные массивы, где порядок следования элементов важен, с помощью этого цикла не стоит.

Пример перебора свойств объекта:

```
var obj = { a: 10, b: 20 };
for (var prop in obj) {
   document.write(prop + " ");
} // a b
```

3.4.5. Цикл *for...of*

Цикл `for...of` используется для перебора объектов, поддерживающих итерации. Он имеет следующий формат:

```
for (<Переменная> of <Объект>) {
   <Тело цикла>
}
```

Цикл `for...of` на каждой итерации присваивает параметру `<Переменная>` значение свойства объекта. Например, с помощью этого цикла можно перебирать элементы массива (листинг 3.19).

Листинг 3.19. Цикл `for...of`

```
var arr = [1, 2, 3];
for (var i of arr) {
   document.write(i + " ");
} // 1 2 3
```

> ***Внимание!***
>
> Цикл `for...of` доступен только в самых последних версиях Web-браузеров. В браузере Internet Explorer в настоящее время его поддержки нет.

3.4.6. Оператор *continue*. Переход на следующую итерацию цикла

Оператор `continue` позволяет перейти на следующую итерацию цикла еще до завершения выполнения всех инструкций внутри цикла. Этот оператор можно применять в любых циклах. Его формат:

```
continue [<Метка>];
```

Выведем все числа от `1` до `100`, кроме чисел от `5` до `10` включительно (листинг 3.20).

Листинг 3.20. Использование оператора `continue`

```
for (var i = 1; i < 101; i++) {
   if (i > 4 && i < 11) continue;
   document.write(i + "<br>");
}
```

3.4.7. Оператор *break*. Прерывание цикла

Оператор `break` позволяет прервать выполнение цикла досрочно. Его формат:

```
break [<Метка>];
```

Выведем все числа от `1` до `100` еще одним способом (листинг 3.21).

Листинг 3.21. Прерывание цикла

```
for (var i = 1; ; i++) {
   if (i > 100) break;
   document.write(i + "<br>");
}
```

Здесь мы не указываем условие продолжения цикла, так что цикл продолжался бы бесконечно, если бы мы не вышли из него, используя оператор `break`.

> ***Внимание!***
>
> Оператор `break` прерывает выполнение цикла, а не программы, т. е. далее будет выполнена инструкция, следующая сразу за циклом.

3.5. Числа

В языке JavaScript для хранения чисел предназначен тип данных `number`:

```
console.log(typeof 1);          // number
console.log(typeof 2.0);        // number
```

Отдельного типа данных для целых чисел в языке JavaScript нет. Числа хранятся в формате вещественных чисел двойной точности.

3.5.1. Указание значений

Целочисленное значение задается в десятичной, двоичной, восьмеричной или шестнадцатеричной форме. Двоичные числа начинаются с комбинации символов `0b` (или `0B`) и могут содержать цифры `0` или `1`. Восьмеричные числа начинаются с нуля и содержат цифры от `0` до `7`. Если встречаются другие цифры, то значение интерпретируется как десятичное. Шестнадцатеричные числа начинаются с комбинации символов `0x` (или `0X`) и могут содержать цифры от `0` до `9` и буквы от `A` до `F` (регистр букв не имеет значения). Двоичные, восьмеричные и шестнадцатеричные значения преобразуются в десятичное значение:

```
// Двоичное значение
console.log(0b01110111);       // 119
// Восьмеричное значение
console.log(0167);             // 119
// Шестнадцатеричное значение
console.log(0x77);             // 119
console.log(0xFF);             // 255
// Десятичное значение
console.log(119);              // 119
```

Вещественное число может содержать точку и (или) экспоненту, начинающуюся с буквы `E` (регистр не имеет значения):

```
console.log(20.0);             // 20
console.log(12.1e20);          // 1.21e+21
console.log(.123);             // 0.123
console.log(47.E-10);          // 4.7e-9
```

При выполнении операций над вещественными числами следует учитывать ограничения точности вычислений. Например, результат следующей инструкции может показаться странным:

```
console.log(0.3 - 0.1 - 0.1 - 0.1);
// -2.7755575615628914e-17
```

Ожидаемым был бы результат `0.0`, но, как видно из примера, мы получили совсем другой результат (`-2.7755575615628914e-17`). Он очень близок к нулю, но не равен нулю. Учитывайте это при указании вещественных чисел в качестве значения счетчика внутри цикла. Попытка проверить такое значение на равенство может привести к бесконечному циклу.

В логическом контексте число `0` и значение `NaN` трактуются как `false`, тогда как любое другое число — как `true`:

```
console.log( !!0 );          // false
console.log( !!0.0 );        // false
console.log( !!NaN );        // false
console.log( !!1 );          // true
console.log( !!-1 );         // true
```

Если выполнить приведение логических значений к типу `number`, то `false` будет приведено к `0`, а `true` — к `1`:

```
console.log( false + 0 );    // 0
console.log( true + 0 );     // 1
```

3.5.2. Класс *Number*

Класс `Number` является классом-оберткой над элементарным типом `number`. Экземпляр класса создается по следующей схеме:

```
<Экземпляр класса> = new Number(<Начальное значение>);
```

Пример создания экземпляра класса:

```
var x = new Number(10);
console.log(x.toString());   // 10
console.log(typeof x);       // object
```

Таким способом редко пользуются. Все методы класса `Number` доступны и при использовании элементарного типа. Преобразование элементарного типа в объектный выполняется автоматически:

```
var x = 255;
console.log(x.toString(16)); // ff
console.log(typeof x);       // number
```

Если в операции участвуют число и экземпляр класса `Number`, то экземпляр класса будет автоматически преобразован в число:

```
var x = new Number(10);
var y = 255;
console.log(x + y);          // 265
```

Для преобразования значения в число можно использовать следующий формат:

```
<Переменная> = Number(<Значение>);
```

Пример преобразования строки в число:

```
var x = Number("10");
console.log(x);              // 10
console.log(typeof x);       // number
```

Свойства класса `Number` можно использовать без создания экземпляра класса:

- `MAX_VALUE` — максимально допустимое в JavaScript число:

  ```
  console.log(Number.MAX_VALUE); // 1.7976931348623157e+308
  ```

- `MIN_VALUE` — минимально допустимое в JavaScript число:

  ```
  console.log(Number.MIN_VALUE); // 5e-324
  ```

- `NaN` — значение `NaN`:

  ```
  console.log(Number.NaN);       // NaN
  ```

- `NEGATIVE_INFINITY` — значение «минус бесконечность»:

  ```
  console.log(Number.NEGATIVE_INFINITY); // -Infinity
  ```

- `POSITIVE_INFINITY` — значение «плюс бесконечность»:

  ```
  console.log(Number.POSITIVE_INFINITY); // Infinity
  ```

- `EPSILON` — значение «эпсилон»:

  ```
  console.log(Number.EPSILON); // 2.220446049250313e-16
  ```

- `MIN_SAFE_INTEGER` — минимальное целочисленное значение:

  ```
  console.log(Number.MIN_SAFE_INTEGER); // -9007199254740991
  ```

- `MAX_SAFE_INTEGER` — максимальное целочисленное значение:

  ```
  console.log(Number.MAX_SAFE_INTEGER); // 9007199254740991
  ```

> ***Примечание***
>
> Свойства `EPSILON`, `MIN_SAFE_INTEGER` и `MAX_SAFE_INTEGER` недоступны в браузере Internet Explorer и доступны только в последних версиях прочих Web-браузеров.

Методы:

- `valueOf()` — возвращает числовое значение экземпляра класса:

  ```
  var x = new Number(15);
  var y = x.valueOf();        // 15
  console.log(typeof y);      // number
  ```

- `toString([<Основание>])` — возвращает строковое представление числа:

  ```
  var x = 15;
  var str = x.toString();     // "15"
  console.log(typeof str);    // string
  x = 100;
  console.log(x.toString(2)); // 1100100
  x = 119;
  ```

```
console.log(x.toString(8));  // 167
x = 255;
console.log(x.toString(16)); // ff
```

- toFixed([<Число>]) — возвращает строковое представление числа, округленное до указанного числа цифр после точки. Если параметр не указан, то значение округляется до целого числа:

```
var x = 1234.6789;
console.log(x.toFixed());          // 1235
console.log(x.toFixed(1));         // 1234.7
console.log(x.toFixed(2));         // 1234.68
```

- toPrecision([<Число>]) — возвращает строковое представление числа, округленное до указанного числа значащих цифр. Если параметр не указан, то метод аналогичен методу toString():

```
var x = 0.06789;
console.log(x.toPrecision());      // 0.06789
console.log(x.toPrecision(1));     // 0.07
console.log(x.toPrecision(2));     // 0.068
```

- toExponential([<Число>]) — возвращает строковое представление числа в экспоненциальной форме, округленное до указанного числа цифр после точки. Если параметр не указан, то округление не выполняется:

```
var x = 0.06789;
console.log(x.toExponential());    // 6.789e-2
console.log(x.toExponential(1));   // 6.8e-2
console.log(x.toExponential(2));   // 6.79e-2
```

- toLocaleString() — возвращает строковое представление числа в зависимости от локали (см. *разд. 3.5.8*):

```
var x = 1027324.564
console.log(x.toLocaleString("ru-RU")); // 1 027 324,564
console.log(x.toLocaleString("en-US")); // 1,027,324.564
console.log(x.toLocaleString("de-DE")); // 1.027.324,564
```

В последних версиях некоторых Web-браузеров доступны также следующие статические методы класса Number:

- isInteger(<Значение>) — возвращает значение true, если указанное значение является целым числом:

```
console.log(Number.isInteger(10));   // true
console.log(Number.isInteger(10.2)); // false
```

- isSafeInteger(<Значение>) — возвращает значение true, если указанное значение является «безопасным» целым числом:

```
console.log(Number.isSafeInteger(10));               // true
console.log(Number.isSafeInteger(
                     Number.MAX_SAFE_INTEGER + 1)); // false
```

- `parseInt(<Строка>[, <Основание>])` — преобразует строку в целое число (аналог глобальной функции `parseInt()`). Строка считается заданной в системе счисления, указанной вторым необязательным параметром. Если основание не указано, то система счисления выбирается автоматически и зависит от префиксов в строке, — например, если строка начинается с `"0x"`, то будет использоваться основание `16`. Если строка не может быть преобразована в число, возвращается значение `NaN`:

```
console.log( Number.parseInt("10") );          // 10
console.log( Number.parseInt("0xFF") );        // 255
console.log( Number.parseInt("0xFF", 16) );    // 255
console.log( Number.parseInt("0167", 8) );     // 119
console.log( Number.parseInt("1110111", 2) );  // 119
console.log( Number.parseInt("строка", 2) );   // NaN
```

- `parseFloat(<Строка>)` — преобразует строку в число с плавающей точкой (аналог глобальной функции `parseFloat()`):

```
console.log( Number.parseFloat("5.2") );       // 5.2
console.log( Number.parseFloat("5e-2") );      // 0.05
console.log( Number.parseFloat("строка") );    // NaN
```

- `isFinite(<Значение>)` — возвращает `true`, если значение не равно плюс или минус бесконечность или значению `NaN`, и `false` — в противном случае (аналог глобальной функции `isFinite()`);
- `isNaN(<Значение>)` — возвращает `true`, если значение равно `NaN`, и `false` — в противном случае (аналог глобальной функции `isNaN()`):

```
var x = 10.0 / 0, y = 0.0 / 0, z = 5;
console.log( x + " " + y + " " + z ); // Infinity NaN 5
console.log( Number.isNaN(x) );       // false
console.log( Number.isNaN(y) );       // true
console.log( Number.isNaN(z) );       // false

console.log( Number.isFinite(x) );    // false
console.log( Number.isFinite(y) );    // false
console.log( Number.isFinite(z) );    // true
```

3.5.3. Математические константы

В классе `Math` определены следующие стандартные константы:

- `PI` — содержит число π:

```
console.log(Math.PI);         // 3.141592653589793
```

- `E` — содержит значение константы `e` (основание натурального логарифма):

```
console.log(Math.E);          // 2.718281828459045
```

- `LN2` — натуральный логарифм из 2:

```
console.log(Math.LN2);        // 0.6931471805599453
```

- `LN10` — натуральный логарифм из 10:

```
console.log(Math.LN10);      // 2.302585092994046
```

- `LOG2E` — логарифм по основанию 2 от `e`:

```
console.log(Math.LOG2E);     // 1.4426950408889634
```

- `LOG10E` — десятичный логарифм от `e`:

```
console.log(Math.LOG10E);    // 0.4342944819032518
```

- `SQRT1_2` — квадратный корень из `0.5`:

```
console.log(Math.SQRT1_2);   // 0.7071067811865476
```

- `SQRT2` — квадратный корень из `2`:

```
console.log(Math.SQRT2);     // 1.4142135623730951
```

3.5.4. Основные методы для работы с числами

В классе `Math` определены следующие основные методы для работы с числами:

- `abs(<Значение>)` — абсолютное значение:

```
console.log(Math.abs(-5));    // 5
```

- `exp(<Значение>)` — экспонента;
- `log(<Значение>)` — натуральный логарифм;
- `pow(<Число>, <Степень>)` — возведение `<Числа>` в `<Степень>`:

```
var x = 5;
console.log(Math.pow(x, 2)); // 25 (5 в квадрате)
```

- `sqrt(<Значение>)` — квадратный корень:

```
var x = 25;
console.log(Math.sqrt(x));    // 5 (квадратный корень из 25)
```

- `max(<Список чисел через запятую>)` — максимальное значение из списка:

```
console.log(Math.max(3, 10, 6)); // 10
```

- `min(<Список чисел через запятую>)` — минимальное значение из списка:

```
console.log(Math.min(3, 10, 6)); // 3
```

3.5.5. Округление чисел

Для округления чисел предназначены следующие методы из класса `Math`:

- `ceil()` — возвращает значение, округленное до ближайшего большего целого:

```
console.log( Math.ceil(1.49) );     // 2
console.log( Math.ceil(1.50) );     // 2
console.log( Math.ceil(1.51) );     // 2
```

- `floor()` — возвращает значение, округленное до ближайшего меньшего целого:

```
console.log( Math.floor(1.49) );   // 1
console.log( Math.floor(1.50) );   // 1
console.log( Math.floor(1.51) );   // 1
```

- `round()` — возвращает число, округленное до ближайшего меньшего целого для чисел с дробной частью, меньшей `0.5`, или значение, округленное до ближайшего большего целого для чисел с дробной частью, большей или равной `0.5`:

```
console.log( Math.round(1.49) );   // 1
console.log( Math.round(1.50) );   // 2
console.log( Math.round(1.51) );   // 2
```

- `trunc()` — отбрасывает дробную часть и возвращает целое число (метод доступен только в последних версиях некоторых Web-браузеров):

```
console.log( Math.trunc(1.5) );    //  1
console.log( Math.trunc(-1.5) );   // -1
console.log( Math.ceil(1.5) );     //  2
console.log( Math.ceil(-1.5) );    // -1
console.log( Math.floor(1.5) );    //  1
console.log( Math.floor(-1.5) );   // -2
```

3.5.6. Тригонометрические функции

В классе `Math` определены следующие основные тригонометрические функции:

- `sin()`, `cos()`, `tan()` — стандартные тригонометрические функции (синус, косинус, тангенс). Значение указывается в радианах:

```
var n = Math.PI / 180;
console.log( Math.sin(90 * n) );    // 1
console.log( Math.cos(10 * n) );    // 0.984807753012208
console.log( Math.tan(45 * n) );    // 0.9999999999999999
```

- `asin()`, `acos()`, `atan()` — обратные тригонометрические функции (арксинус, арккосинус, арктангенс). Значение возвращается в радианах.

3.5.7. Преобразование строки в число

Числа, вводимые пользователем, например, в поле диалогового окна, представлены в виде строки. Чтобы в дальнейшем использовать эти числа, необходимо выполнить преобразование строки в число. Для этого предназначены следующие функции:

- `parseInt(<Строка>[, <Основание>])` — преобразует строку в целое число. Строка считается заданной в системе счисления, указанной вторым необязательным параметром. Если основание не указано, то система счисления выбирается автоматически и зависит от префиксов в строке, — например, если строка начинается с `"0x"`, то будет использоваться основание `16`. Если строка не может быть преобразована в число, возвращается значение `NaN`:

```
console.log( parseInt("10") );          // 10
console.log( parseInt("0xFF") );        // 255
console.log( parseInt("0xFF", 16) );    // 255
console.log( parseInt("0167", 8) );     // 119
console.log( parseInt("1110111", 2) ); // 119
console.log( parseInt("строка", 2) );  // NaN
```

- `parseFloat(<Строка>)` — преобразует строку в число с плавающей точкой:

```
console.log( parseFloat("5.2") );       // 5.2
console.log( parseFloat("5e-2") );      // 0.05
console.log( parseFloat("строка") );    // NaN
```

В последних версиях некоторых Web-браузеров доступны также статические методы `parseInt()` и `parseFloat()` из класса `Number` (см. *разд. 3.5.2*):

```
console.log( Number.parseInt("0xFF") );      // 255
console.log( Number.parseFloat("5.2") );     // 5.2
```

3.5.8. Преобразование числа в строку

Чтобы преобразовать число в строку, достаточно выполнить операцию конкатенации:

```
console.log( 10 + " " + 1.2);         // 10 1.2
```

Для преобразования числа в строку можно также воспользоваться методом `toString([<Основание>])`:

```
var x = 10, y = 5.2;
console.log( x.toString() );     // 10
console.log( x.toString(8) );    // 12
console.log( y.toString() );     // 5.2
```

Метод `toLocaleString([<Локаль>[, <Настройки>]])` возвращает строковое представление числа в зависимости от локали. В первом параметре можно задать название локали:

```
var x = 1027324.5647
console.log(x.toLocaleString("ru-RU")); // 1 027 324,565
console.log(x.toLocaleString("en-US")); // 1,027,324.565
console.log(x.toLocaleString("de-DE")); // 1.027.324,565
```

Во втором параметре указывается объект с дополнительными настройками:

- `style` — стиль форматирования: "`decimal`" (просто число, значение по умолчанию), "`currency`" (валюта) или "`percent`" (проценты);
- `currency` — задает код валюты для стиля "`currency`";
- `currencyDisplay` — задает способ отображения валюты для стиля "`currency`": "`symbol`" (символ валюты, значение по умолчанию), "`code`" (код валюты, например "`RUB`") или "`name`" (название валюты, например "`евро`"):

```
var x = 1027324.5647
console.log( x.toLocaleString("ru-RU",
           { style: "currency", currency: "RUB",
             currencyDisplay: "code"})); // 1 027 324,56 RUB
```

- `minimumIntegerDigits` — минимальное количество цифр в целой части числа (значение по умолчанию: `1`);
- `minimumFractionDigits` — минимальное количество цифр в дробной части числа (значение по умолчанию: `0`);
- `maximumFractionDigits` — максимальное количество цифр в дробной части числа:

```
var x = 1027324.5647;
console.log( x.toLocaleString("ru-RU",
           { style: "decimal", minimumIntegerDigits: 1,
             minimumFractionDigits: 0,
             maximumFractionDigits: 2 })); // 1 027 324,56
```

- `useGrouping` — определяет, нужно ли выводить разделители тысячных групп. Можно указать значения `true` (разделители выводятся, значение по умолчанию) или `false` (разделители не выводятся):

```
var x = 1027324.5647;
console.log( x.toLocaleString("ru-RU",
           { style: "decimal", useGrouping: false,
             maximumFractionDigits: 2 })); // 1027324,56
```

Вместо метода `toLocaleString()` можно воспользоваться классом `Intl.NumberFormat`:

```
<Экземпляр класса> = new Intl.NumberFormat([<Локаль>[, <Настройки>]]);
```

Применить настройки и получить отформатированную строку позволяет метод `format(<Число>)`:

```
var x = 1027324.5647;
var obj = new Intl.NumberFormat("ru-RU");
console.log(obj.format(x));                     // 1 027 324,565
obj = new Intl.NumberFormat("ru-RU",
               { style: "decimal", useGrouping: false,
                 maximumFractionDigits: 2 });
console.log(obj.format(x));                     // 1027324,56
```

Метод `resolvedOptions()` возвращает объект со всеми настройками:

```
var obj = new Intl.NumberFormat("ru-RU");
console.log(obj.resolvedOptions());
// Object { locale: "ru-RU", numberingSystem: "latn",
// style: "decimal", minimumIntegerDigits: 1, minimumFractionDigits: 0,
// maximumFractionDigits: 3, useGrouping: true }
```

В последних версиях некоторых Web-браузеров доступны также методы `toFixed()`, `toPrecision()` и `toExponential()` из класса `Number` (см. *разд. 3.5.2*):

```
var x = 0.06789;
console.log(x.toFixed(2));         // 0.07
console.log(x.toPrecision(2));     // 0.068
console.log(x.toExponential(2));   // 6.79e-2
```

3.5.9. Генерация псевдослучайных чисел

Для генерации псевдослучайных чисел предназначен метод `random()` из класса `Math`. Он возвращает псевдослучайное число от `0` до `1`:

```
console.log(Math.random()); // например, 0.6748229072896632
```

Для того чтобы получить случайное целое число от `0` до `9`, нужно возвращаемое методом `random()` значение умножить на `9.9999`, а затем округлить число до ближайшего меньшего целого при помощи метода `floor()`:

```
console.log(Math.floor(Math.random() * 9.9999)); // например, 2
```

Попробуйте несколько раз обновить Web-страницу. Число будет меняться случайным образом в пределах от `0` до `9` включительно. Для чего это может пригодиться? Например, если есть четыре баннера 468×60, то их можно показывать случайным способом:

```
var x = Math.floor(Math.random() * 3.9999);
document.write('<img src="banner' + x +
               '.gif" width="480" height="60" alt="">');
```

Файлы четырех баннеров с именами banner0.gif, banner1.gif, banner2.gif и banner3.gif должны быть расположены в одной папке с файлом, в котором находится исполняемый скрипт.

Названия файлов с баннерами можно сделать произвольными, добавив их в массив:

```
var arr = [ "banner-red.gif", "banner-blue.jpeg",
            "banner-gray.gif", "banner-white.png" ];
var x = Math.floor(Math.random() * 3.9999);
document.write('<img src="banner' + arr[x] +
               '.gif" width="480" height="60" alt="">');
```

3.5.10. Бесконечность и значение *NaN*

Деление числа на `0` приведет к значению плюс или минус `Infinity` (бесконечность), а деление числа `0.0` на `0` — к значению `NaN` (нет числа):

```
console.log(  10 / 0 );            // Infinity
console.log( -10 / 0 );            // -Infinity
console.log(   0 / 0 );            // NaN
```

Присвоить эти значения можно также с помощью глобальных свойств `Infinity` и `NaN` или свойств `POSITIVE_INFINITY`, `NEGATIVE_INFINITY` и `NaN` из класса `Number`:

```
console.log( Infinity );           // Infinity
console.log( -Infinity );          // -Infinity
```

```
console.log( NaN );                         // NaN
console.log(Number.POSITIVE_INFINITY); // Infinity
console.log(Number.NEGATIVE_INFINITY); // -Infinity
console.log(Number.NaN);                // NaN
```

Любая операция значения `Infinity` с числом даст в результате `Infinity`:

```
console.log( Infinity - 5 );            // Infinity
```

Любая операция со значением `NaN` даст в результате `NaN`:

```
console.log( NaN + 5 );                 // NaN
```

При сравнении значение `NaN` не равно даже самому себе:

```
console.log( NaN == NaN );              // false
console.log( NaN === NaN );             // false
```

Для проверки соответствия этим значениям следует воспользоваться следующими глобальными функциями:

- `isFinite(<Значение>)` — возвращает `true`, если значение не равно плюс или минус бесконечность или значению `NaN`, и `false` — в противном случае;
- `isNaN(<Значение>)` — возвращает `true`, если значение равно `NaN`, и `false` — в противном случае:

```
var x = 10.0 / 0, y = 0.0 / 0, z = 5;
console.log( x + " " + y + " " + z ); // Infinity NaN 5
console.log( isNaN(x) );              // false
console.log( isNaN(y) );              // true
console.log( isNaN(z) );              // false

console.log( isFinite(x) );           // false
console.log( isFinite(y) );           // false
console.log( isFinite(z) );           // true
```

В последних версиях некоторых Web-браузеров доступны также статические методы `isFinite()` и `isNaN()` из класса `Number` (см. *разд. 3.5.2*):

```
var x = 10.0 / 0, y = 0.0 / 0, z = 5;
console.log( Number.isNaN(y) );       // true
console.log( Number.isFinite(x) );    // false
console.log( Number.isFinite(z) );    // true
```

3.6. Массивы и множества

Массив — это нумерованный набор переменных. Переменная в массиве называется *элементом массива*, а ее позиция в массиве задается *индексом*. Нумерация элементов массива начинается с `0`, а не с `1`. Это следует помнить. Общее количество элементов в массиве называется *размером массива*. Массивы, ключами которых являются числа, часто называют *списками*, а массивы, ключами которых являются строки, — *ассоциативными массивами*.

3.6.1. Инициализация массива

Массив инициализируют следующими способами:

- переменные указываются через запятую в квадратных скобках:

```
var arr = [1, 2, 3, 4];
console.log( arr );             // Array [ 1, 2, 3, 4 ]
console.log( typeof arr );      // object
```

- с помощью класса `Array`. Класс предоставляет доступ к множеству методов для обработки массивов. Экземпляр класса можно создать следующими способами:

```
<Экземпляр класса> = new Array(<Количество элементов массива>);
<Экземпляр класса> = new Array(
                         [<Элементы массива через запятую>]);
```

  Если в круглых скобках нет никаких параметров, то создается массив нулевой длины, т. е. массив, не содержащий элементов:

```
var arr = new Array();
console.log( arr );             // Array [  ]
console.log( typeof arr );      // object
```

  Если указано одно число, то это число задает количество элементов массива:

```
var arr = new Array(3);
console.log( arr );             // Array [ <3 пустых элемента> ]
```

  Если указано несколько элементов через запятую, или единственное значение не является числом, то указанные значения записываются в создаваемый массив:

```
var arr = new Array(1, 2, 3, 4);
console.log( arr );             // Array [ 1, 2, 3, 4 ]
```

- с помощью статического метода `of(<Список значений>)` из класса `Array`. Метод доступен только в последних версиях некоторых Web-браузеров:

```
var arr = Array.of(1, 2, 3, 4);
console.log(arr);  // Array [ 1, 2, 3, 4 ]
```

Создать массив на основе другого объекта, поддерживающего итерации, можно с помощью статического метода `from()` из класса `Array`:

```
Array.from(<Объект>[, <Функция>[, <this>]])
```

В первом параметре указывается объект, элементы которого будут преобразованы в массив. Во втором параметре можно указать функцию, которая будет вызвана для каждого элемента. Объект, указанный в третьем параметре, будет доступен внутри функции через указатель `this`:

```
console.log(Array.from( "123" ));  // Array [ "1", "2", "3" ]
console.log(Array.from( [ 1, 2, 3 ], function(n) {
   return n * 2;
}));                               // Array [ 2, 4, 6 ]
```

Обратите внимание: этот метод доступен только в последних версиях некоторых Web-браузеров.

Проверить, является ли объект массивом, позволяет статический метод `isArray()` из класса `Array`. Метод возвращает значение `true`, если объект является массивом, и `false` — в противном случае:

```
console.log(Array.isArray( [1, 2] ));  // true
console.log(Array.isArray( {} ));      // false
```

3.6.2. Получение и изменение элемента массива

Получить значение элемента массива можно, указав его индекс в квадратных скобках. Нумерация элементов массива начинается с нуля, а не с единицы:

```
var arr = [1, 2, 3];
console.log( arr[0] );   // 1
console.log( arr[1] );   // 2
console.log( arr[2] );   // 3
```

Если элемент с указанным индексом не существует, то будет возвращено значение `undefined`:

```
console.log( arr[4] ); // undefined
```

При желании можно добавить новый элемент массива или изменить значение существующего:

```
var arr = [1, 2];
arr[2] = 3;          // Добавление нового элемента
arr[0] = 55;         // Изменение значения существующего элемента
console.log( arr ); // Array [ 55, 2, 3 ]
```

Если при добавлении указать индекс, превышающий количество элементов массива, то будет создан элемент с указанным индексом, а также промежуточные элементы со значением `undefined`:

```
var arr = [1, 2];
arr[5] = 3;          // Добавление нового элемента
console.log( arr ); // Array [ 1, 2, <3 пустых элемента>, 3 ]
```

В последних версиях некоторых Web-браузеров существует метод `fill()`, с помощью которого можно заполнить весь массив или только диапазон одинаковым значением. Формат метода:

```
fill(<Значение>[, <Начало>[, <Конец>]])
```

В первом параметре указывается вставляемое значение. Если остальные параметры не заданы, то это значение заполнит весь массив. В параметре `<Начало>` можно указать начальный индекс диапазона (значение по умолчанию: `0`), а в параметре `<Конец>` — конечный индекс (значение по умолчанию: число элементов массива):

```
var arr = new Array(5);
arr.fill(1);
```

```
console.log(arr); // Array [ 1, 1, 1, 1, 1 ]
arr.fill(2, 1, 4);
console.log(arr); // Array [ 1, 2, 2, 2, 1 ]
```

3.6.3. Определение числа элементов массива

Получить число элементов массива позволяет свойство `length`:

```
var arr = [ "Один", "Два", "Три" ];
console.log(arr.length); // 3
```

3.6.4. Многомерные массивы

Многомерные массивы можно создать перечислением:

```
var arr = new Array(new Array("Один", "Два", "Три"),
                    new Array("Четыре", "Пять", "Шесть"));
console.log(arr[0][1]); // Два
var arr2 = [
              [ "Один", "Два", "Три" ],
              [ "Четыре", "Пять", "Шесть" ]
           ];
console.log(arr2[1][1]); // Пять
```

или поэлементно:

```
var arr = new Array();
arr[0] = new Array();
arr[1] = new Array();
arr[0][0] = "Один";
arr[0][1] = "Два";
arr[0][2] = "Три";
arr[1][0] = "Четыре";
arr[1][1] = "Пять";
arr[1][2] = "Шесть";
console.log(arr[1][2]); // Шесть
var arr2 = [];
arr2[0] = [];
arr2[1] = [];
arr2[0][0] = "Один";
arr2[0][1] = "Два";
arr2[0][2] = "Три";
arr2[1][0] = "Четыре";
arr2[1][1] = "Пять";
arr2[1][2] = "Шесть";
console.log(arr2[0][0]); // Один
```

Обращение к элементу многомерного массива осуществляется с помощью двух индексов:

```
console.log(arr[1][2]); // Шесть
```

Любому элементу массива можно присвоить другой массив, что позволяет создавать так называемые «зубчатые» многомерные массивы:

```
var arr = [1, 2, 3];
arr[1] = [2, 3];
arr[2] = [4, 5, 6];
for (var i = 0; i < arr.length; i++) {
   console.log(arr[i]);
}
```

Результат:

```
1
Array [ 2, 3 ]
Array [ 4, 5, 6 ]
```

3.6.5. Создание копии массива

Следует учитывать, что операция присваивания сохраняет в переменной ссылку на массив, а не все его значения. Например, если попробовать сделать так, то изменение `arr2` затронет `arr1`:

```
var arr1, arr2;
arr1 = [1, 2, 3, 4];
arr2 = arr1;          // Присваивается ссылка на массив!!!
arr2[0] = 88;
console.log(arr1);    // Array [ 88, 2, 3, 4 ]
console.log(arr2);    // Array [ 88, 2, 3, 4 ]
```

Чтобы сделать копию массива, можно, например, воспользоваться методом `slice()`, который возвращает срез массива:

```
var arr1, arr2;
arr1 = [1, 2, 3, 4];
arr2 = arr1.slice(0);
arr2[0] = 88;
console.log(arr1);    // Array [ 1, 2, 3, 4 ]
console.log(arr2);    // Array [ 88, 2, 3, 4 ]
```

Необходимо заметить, что при использовании многомерных массивов метод `slice()` создает «поверхностную» копию, а не полную:

```
var arr1, arr2;
arr1 = [ [0, 1], 2, 3, 4 ];
arr2 = arr1.slice(0);
arr2[0][0] = 55;
arr2[1] = 88;
console.log(arr1[0]);  // Array [ 55, 1 ]
console.log(arr2[0]);  // Array [ 55, 1 ]
console.log(arr1[1]);  // 2
console.log(arr2[1]);  // 88
```

Как видно из примера, изменение вложенного массива в `arr2` привело к одновременному изменению значения в `arr1`. Иными словами, оба массива содержат ссылку на один и тот же вложенный массив.

В последних версиях некоторых Web-браузеров для создания «поверхностной» копии массива можно воспользоваться статическим методом `from()` из класса `Array`:

```
var arr1 = [1, 2, 3, 4], arr2;
arr2 = Array.from(arr1);
arr2[0] = 88;
console.log(arr1);    // Array [ 1, 2, 3, 4 ]
console.log(arr2);    // Array [ 88, 2, 3, 4 ]
```

3.6.6. Слияние массивов

Метод `concat(<Список элементов или массивов>)` возвращает массив, полученный в результате объединения текущего массива и списка элементов или других массивов. При этом в текущий массив элементы не добавляются:

```
var arr = [1, 2], arr2 = [3, 4], arr3 = [];
arr3 = arr.concat(arr2);
console.log(arr3);          // Array [ 1, 2, 3, 4 ]
arr3 = arr.concat(arr2, 5, 6);
console.log(arr3);          // Array [ 1, 2, 3, 4, 5, 6 ]
```

3.6.7. Перебор элементов массива

Для перебора массивов лучше всего подходит цикл `for`. Умножим все элементы массива на `2`:

```
var arr = [1, 2, 3, 4];
for (var i = 0; i < arr.length; i++) {
   arr[i] *= 2;
}
console.log(arr); // Array [ 2, 4, 6, 8 ]
```

В последних версиях Web-браузеров можно также воспользоваться методом `forEach()`. Формат метода:

```
forEach(<Функция>[, <this>])
```

В первом параметре указывается ссылка на функцию, которая будет вызвана для каждого элемента массива, не имеющего значение `undefined`. Функция принимает три параметра: через первый параметр доступно значение текущего элемента, через второй — его индекс, а через третий — ссылка на массив. Значение, указанное в параметре `<this>`, будет доступно через указатель `this` внутри функции.

Прибавим ко всем элементам значение `2`:

```
var arr = [1, 2, 3, 4];
arr.forEach(function(value, index, ar) {
```

```
   ar[index] = value + 2;
});
console.log(arr); // Array [ 3, 4, 5, 6 ]
```

Перебрать элементы массива в последних версиях Web-браузеров позволяет также цикл `for...of`. Он имеет следующий формат:

```
for (<Переменная> of <Массив>) {
   <Тело цикла>
}
```

Цикл `for...of` на каждой итерации присваивает параметру `<Переменная>` копию значения элемента массива. Посчитаем сумму всех элементов массива:

```
var arr = [1, 2, 3], n = 0;
for (var i of arr) {
   n += i;
}
console.log(n); // 6
```

3.6.8. Добавление и удаление элементов массива

Добавить новый элемент массива или изменить значение существующего можно с помощью квадратных скобок:

```
var arr = [1, 2, 3];
arr[3] = 4;
arr[arr.length] = 5;  // Добавление в конец массива
console.log(arr);     // Array [ 1, 2, 3, 4, 5 ]
```

Если при добавлении указать индекс, превышающий количество элементов массива, то будет создан элемент с указанным индексом, а также промежуточные элементы со значением `undefined`:

```
var arr = [1, 2];
arr[5] = 3;          // Добавление нового элемента
console.log( arr ); // Array [ 1, 2, <3 пустых элемента>, 3 ]
```

Для добавления и удаления элементов можно воспользоваться следующими методами:

- `push(<Список элементов>)` — добавляет в массив элементы, указанные в списке элементов. Элементы добавляются в конец массива. Метод возвращает новую длину массива:

  ```
  var arr = [1, 2, 3];
  console.log(arr.push(4, 5));    // 5
  console.log(arr);               // Array [ 1, 2, 3, 4, 5 ]
  ```

  Для добавления элементов из какого-либо массива в конец текущего массива можно воспользоваться следующим кодом:

  ```
  var arr = [1, 2, 3], arr2 = [4, 5];
  Array.prototype.push.apply(arr, arr2);
  console.log(arr);               // Array [ 1, 2, 3, 4, 5 ]
  ```

- `unshift(<Список элементов>)` — добавляет в массив элементы, указанные в списке элементов. Элементы добавляются в начало массива. Метод возвращает новую длину массива:

```
var arr = [1, 2, 3];
console.log(arr.unshift(4, 5)); // 5
console.log(arr);               // Array [ 4, 5, 1, 2, 3 ]
```

- `shift()` — удаляет первый элемент массива и возвращает его (или значение `undefined`, если элементов больше нет):

```
var arr = [1, 2, 3];
console.log(arr.shift());       // 1
console.log(arr);               // Array [ 2, 3 ]
```

- `pop()` — удаляет последний элемент массива и возвращает его (или значение `undefined`, если элементов больше нет):

```
var arr = [1, 2, 3];
console.log(arr.pop());         // 3
console.log(arr);               // Array [ 1, 2 ]
```

- `splice(<Начало>, <Количество>[, <Список значений>])` — позволяет удалить, заменить или вставить элементы массива. Возвращает массив, состоящий из удаленных элементов:

```
var arr1 = [ 1, 2, 3, 4, 5 ];
var arr2 = arr1.splice(2, 2);
console.log(arr1);              // Array [ 1, 2, 5 ]
console.log(arr2);              // Array [ 3, 4 ]
var arr3 = arr1.splice(1, 1, 7, 8, 9);
console.log(arr1);              // Array [ 1, 7, 8, 9, 5 ]
console.log(arr3);              // Array [ 2 ]
var arr4 = arr1.splice(1, 0, 2, 3, 4);
console.log(arr1);          // Array [ 1, 2, 3, 4, 7, 8, 9, 5 ]
console.log(arr4);              // Array [  ]
```

3.6.9. Переворачивание массива

Метод `reverse()` переворачивает массив — его элементы будут следовать в обратном порядке относительно исходного массива:

```
var arr = [ 1, 2, 3 ];
arr.reverse();
console.log(arr);               // Array [ 3, 2, 1 ]
```

3.6.10. Сортировка массива

Отсортировать массив позволяет метод `sort([Функция сортировки])`. Если функция сортировки не указана, будет выполнена обычная сортировка (числа сортируются по возрастанию, а символы — по алфавиту):

```
var arr = [ "Один", "Два", "Три" ];
arr.sort();
console.log(arr); // Array [ "Два", "Один", "Три" ]
```

Если нужно изменить стандартный порядок сортировки, это можно сделать с помощью пользовательской функции сортировки. Функция принимает две переменные и должна возвращать:

- `положительное число` — если первый элемент больше второго;
- `отрицательное число` — если второй элемент больше первого;
- `0` — если элементы равны.

Например, стандартная сортировка зависит от регистра символов:

```
var arr = [ "единица1", "Единый", "Единица2" ];
arr.sort();
console.log(arr); // Array [ "Единица2", "Единый", "единица1" ]
```

В результате мы получим неправильную сортировку, ведь `"Единица2"` и `"Единый"` должны стоять позже `"единица1"`. Изменим стандартную сортировку на свою сортировку без учета регистра (листинг 3.22).

Листинг 3.22. Сортировка без учета регистра

```
function cmp(str1, str2) {      // Сортировка без учета регистра
   str1 = str1.toLowerCase();   // Преобразуем к нижнему регистру
   str2 = str2.toLowerCase();   // Преобразуем к нижнему регистру
   if (str1 > str2) return 1;
   if (str1 < str2) return -1;
   return 0;
}
var arr = [ "единица1", "Единый", "Единица2" ];
arr.sort(cmp);      // Имя функции указывается без скобок
console.log(arr);   // Array [ "единица1", "Единица2", "Единый" ]
```

Для этого две переменные приводим к одному регистру, а затем производим стандартное сравнение. Обратите внимание, что мы не изменяем регистр самих элементов массива, т. к. работаем с их копиями.

Порядок сортировки можно изменить на противоположный (листинг 3.23), изменив возвращаемые функцией значения. В качестве примера укажем анонимную функцию вместо обычной.

Листинг 3.23. Сортировка без учета регистра в обратном порядке

```
var arr = [ "единица1", "Единый", "Единица2" ];
arr.sort(function(str1, str2) {
   str1 = str1.toLowerCase();
   str2 = str2.toLowerCase();
```

```
    if (str1 > str2) return -1;
    if (str1 < str2) return 1;
    return 0;
});
console.log(arr);  // Array [ "Единый", "Единица2", "единица1" ]
```

3.6.11. Получение части массива

Метод `slice(<Начало>[, <Конец>])` возвращает срез массива, начиная от индекса `<Начало>` и заканчивая индексом `<Конец>`, но не включает элемент с этим индексом. Если второй параметр не указан, то возвращаются все элементы до конца массива:

```
var arr1 = [ 1, 2, 3, 4, 5 ];
var arr2 = arr1.slice(1, 4);
console.log(arr2);              // Array [ 2, 3, 4 ]
var arr3 = arr1.slice(2);
console.log(arr3);              // Array [ 3, 4, 5 ]
```

В первом параметре можно указать отрицательное значение. В этом случае смещение отсчитывается от конца массива. Получим два последних элемента:

```
var arr1 = [ 1, 2, 3, 4, 5 ];
var arr2 = arr1.slice(-2);
console.log(arr2);              // Array [ 4, 5 ]
```

3.6.12. Преобразование массива в строку

Преобразовать массив в строку позволяют следующие методы:

- `join([<Разделитель>])` — возвращает строку, полученную в результате объединения всех элементов массива через разделитель:

  ```
  var arr = [ "Один", "Два", "Три" ];
  var str = arr.join(" - ");
  console.log(str); // Один - Два - Три
  ```

  Если параметр не указан, то в качестве символа-разделителя будет использоваться запятая:

  ```
  var arr = [ "Один", "Два", "Три" ];
  var str = arr.join();
  console.log(str); // Один,Два,Три
  ```

- `toString()` — преобразует массив в строку. Элементы указываются через запятую без пробела. Метод вызывается автоматически при использовании строкового контекста:

  ```
  var arr = [ "Один", "Два", "Три" ];
  console.log(arr.toString()); // Один,Два,Три
  console.log(arr + "");       // Один,Два,Три
  ```

- `toLocaleString()` — преобразует массив в строку. Элементы указываются через запятую без пробела. Если элементами массива являются объекты, то для преобразования в строку будет вызван метод `toLocaleString()` этого объекта:

```
var arr = [ new Date(), 1234.45678 ];
console.log(arr.toString());
// Wed Dec 06 2017 11:10:58 GMT+0300,1234.45678
console.log(arr.toLocaleString());
// 06.12.2017, 11:10:58,1 234,457
```

3.6.13. Проверка наличия элемента в массиве

Выполнить поиск элемента в массиве позволяют следующие методы:

- `indexOf(<Элемент>[, <Индекс>])` — ищет первое вхождение элемента в массиве. Сравнение выполняется с помощью оператора `===` (строго равно). Возвращает индекс найденного элемента или значение `-1`, если элемент не найден. Во втором параметре можно указать индекс, с которого начнется поиск. Если второй параметр не указан, то поиск начинается с начала массива:

```
var arr = [ 1, 2, 1, 2, 3 ];
console.log(arr.indexOf(2));        // 1
console.log(arr.indexOf(2, 2));     // 3
console.log(arr.indexOf(3));        // 4
console.log(arr.indexOf(8));        // -1
```

- `lastIndexOf(<Элемент>[, <Индекс>])` — ищет последнее вхождение элемента в массиве:

```
var arr = [ 1, 2, 1, 2, 3 ];
console.log(arr.lastIndexOf(2));    // 3
console.log(arr.lastIndexOf(2, 2)); // 1
console.log(arr.lastIndexOf(3));    // 4
console.log(arr.lastIndexOf(8));    // -1
```

- `includes(<Элемент>[, <Индекс>])` — возвращает значение `true`, если элемент присутствует в массиве, и `false` — в противном случае. Во втором параметре можно указать индекс элемента, с которого начнется поиск:

```
var arr = [ 1, 2, 3 ], arr2 = [ 1, 2, -3 ];
console.log(arr.includes(1));    // true
console.log(arr.includes(1, 1)); // false
console.log(arr2.includes(3));   // false
```

- `every(<Функция>[, <this>])` — возвращает значение `true`, если все элементы массива соответствуют заданному условию, и `false` — в противном случае. В первом параметре указывается ссылка на функцию, которая будет вызвана для каждого элемента массива. Функция принимает три параметра: через первый параметр доступно значение текущего элемента, через второй — его индекс, а через третий — ссылка на массив. Функция должна вернуть значение `true`, если элемент соответствует условию, и `false` — в противном случае. Если функция для всех элементов вернула `true`, то метод `every()` также возвращает значение

`true`. Если функция вернет значение `false`, то проверка прекращается, и метод `every()` возвращает значение `false`. Значение, указанное во втором параметре, будет доступно внутри функции через указатель `this`:

```
function test(value, index, ar) {
   return value > 0;
}
var arr = [ 1, 2, 3 ], arr2 = [ 1, 2, -3 ];
console.log(arr.every(test));    // true
console.log(arr2.every(test));   // false
```

- `some(<Функция>[, <this>])` — возвращает значение `true`, если хотя бы один элемент массива соответствует заданному условию, и `false` — в противном случае. Если функция вернет значение `true`, то проверка прекращается, и метод `some()` возвращает значение `true`. Если функция для всех элементов вернула `false`, то метод `some()` также возвращает значение `false`:

```
function test(value, index, ar) {
   return value < 0;
}
var arr = [ 1, 2, 3 ], arr2 = [ 1, 2, -3 ];
console.log(arr.some(test));    // false
console.log(arr2.some(test));   // true
```

- `find(<Функция>[, <this>])` — возвращает значение первого элемента, для которого функция вернула `true`, или значение `undefined`, если функция для всех элементов вернула `false`:

```
function test(value, index, ar) {
   return value < 0;
}
var arr = [ 1, 2, 3 ], arr2 = [ 1, 2, -3 ];
console.log(arr.find(test));    // undefined
console.log(arr2.find(test));   // -3
```

- `findIndex(<Функция>[, <this>])` — возвращает индекс первого элемента, для которого функция вернула `true`, или значение `-1`, если функция для всех элементов вернула `false`:

```
function test(value, index, ar) {
   return value < 0;
}
var arr = [ 1, 2, 3 ], arr2 = [ 1, 2, -3 ];
console.log(arr.findIndex(test));    // -1
console.log(arr2.findIndex(test));   // 2
```

3.6.14. Фильтрация массива

Метод `filter(<Функция>[, <this>])` позволяет выполнить фильтрацию массива. В первом параметре указывается ссылка на функцию, которая будет вызвана для каждого элемента массива. Функция принимает три параметра: через первый пара-

метр доступно значение текущего элемента, через второй — его индекс, а через третий — ссылка на массив. Функция должна вернуть значение `true`, если элемент соответствует условию, и `false` — в противном случае. Значение, указанное во втором параметре, будет доступно внутри функции через указатель `this`. Метод возвращает массив элементов, соответствующих условию (для которых функция вернула значение `true`). Исходный массив не изменяется:

```
function test(value, index, ar) {
   return value < 4;
}
var arr = [ 1, 2, 3, 4, 5, 6 ];
var arr2 = arr.filter(test);
console.log(arr2);                    // Array [ 1, 2, 3 ]
```

С помощью метода `map(<Функция>[, <this>])` можно применить пользовательскую функцию ко всем элементам массива. Внутри функции нужно вернуть новое значение элемента. Метод `map()` возвращает отфильтрованный массив.

Вот пример умножения всех элементов массива на `2`:

```
var arr = [ 1, 2, 3, 4, 5, 6 ];
var arr2 = arr.map(function(value, index, ar) {
   return value * 2;
});
console.log(arr2);                // Array [ 2, 4, 6, 8, 10, 12 ]
```

Метод `reduce(<Функция>[, <Начальное значение>])` применяет функцию к парам элементов и накапливает результат. В первом параметре указывается ссылка на функцию, которая будет вызвана для каждого элемента массива. Функция принимает четыре параметра: через первый параметр доступен результат предыдущих вычислений или `<Начальное значение>` при первом вызове, через второй — значение текущего элемента, через третий — его индекс, а через четвертый — ссылка на массив. Внутри функции нужно вернуть результат текущих вычислений. Метод `reduce()` возвращает общий результат вычислений.

Вот пример получения суммы всех элементов массива:

```
var arr = [ 1, 2, 3 ];
var sum = arr.reduce(function(value1, value2, index, ar) {
   return value1 + value2;
}, 0);
console.log(sum);                // 6
```

Метод `reduceRight()` аналогичен методу `reduce()`, но перебирает элементы не слева направо, а справа налево.

Вот пример преобразования многомерного массива в одномерный:

```
var arr = [ [1, 2], [3, 4] ];
var sum = arr.reduceRight(function(arr1, arr2, index, ar) {
   return arr1.concat(arr2);
}, []);
console.log(sum);                // Array [ 3, 4, 1, 2 ]
```

3.6.15. Ассоциативные массивы

Основным отличием ассоциативных массивов от обычных является возможность обращения к элементу массива не по числовому индексу, а по *ключу*, представляющему собой строку. Ключи ассоциативных массив в JavaScript на самом деле являются свойствами объекта, поэтому для создания ассоциативного массива лучше использовать класс `Object`:

```
var arr = new Object();
arr["Один"] = 1;
arr["Два"] = 2;
arr["Три"] = 3;
console.log(arr["Один"]); // 1
console.log(arr["Два"]);  // 2
console.log(arr["Три"]);  // 3
```

Ассоциативные массивы используются также для доступа к свойствам объекта вместо классической точки. Для получения числа элементов массива ранее мы обращались к свойству `length` класса `Array` следующим образом:

```
var arr = [ "Один", "Два", "Три" ];
console.log(arr.length); // 3
```

С помощью ассоциативных массивов обращение к свойству `length` будет выглядеть так:

```
var arr = [ "Один", "Два", "Три" ];
console.log(arr["length"]); // 3
```

Перебор ассоциативных массивов

Как вывести все элементы массива? Ни один из методов класса `Array` не позволяет вывести элементы ассоциативного массива. Кстати, свойство `length` также не работает. По этой причине перебрать все элементы массива с помощью стандартного цикла `for` не получится.

Для этой цели существует специальный цикл `for...in`. Он имеет следующий формат:

```
for (<Переменная> in <Экземпляр класса>) {
   <Тело цикла>
}
```

Цикл `for...in` на каждой итерации присваивает параметру `<Переменная>` имя свойства (ключа), с помощью которого можно получить значение соответствующего элемента ассоциативного массива:

```
var arr = new Object();
arr["Один"] = 1;
arr["Два"] = 2;
arr["Три"] = 3;
```

```
for (var key in arr) {
   // Переменной key на каждой итерации присваивается
   // ключ текущего элемента ассоциативного массива
   console.log(key + " = " + arr[key]);
}
```

В итоге мы получим следующий результат:

```
Один = 1
Два = 2
Три = 3
```

Класс *Map*

В последних версиях некоторых Web-браузеров для создания ассоциативного массива можно воспользоваться классом `Map`. В отличие от обычных свойств объектов, ключами могут быть не только строки, но и другие объекты. Пример использования класса `Map` приведен в листинге 3.24.

Листинг 3.24. Класс `Map`

```
var arr = new Map();
// Добавление элементов и изменение значений
arr.set("Один", 1);
arr.set("Два", 2);
arr.set("Три", 3);
// Получение элемента по ключу
console.log(arr.get("Два")); // 2
// Получение числа элементов
console.log(arr.size);       // 3
// Проверка наличия ключа
console.log(arr.has("Три")); // true
console.log(arr.has("Сто")); // false
// Получение списка ключей и значений
console.log([...arr.keys()]);   // Array [ "Один", "Два", "Три" ]
console.log([...arr.values()]); // Array [ 1, 2, 3 ]
// Перебор элементов
for (var key of arr.keys()) {
   console.log(key + " - " + arr.get(key));
}
arr.forEach(function(value, key, ar) {
   console.log(key + " - " + value);
});
console.log(arr); // Map { Один: 1, Два: 2, Три: 3 }
// Удаление элементов
console.log(arr.delete("Три")); // true
console.log(arr.size);          // 2
arr.clear();
console.log(arr.size);          // 0
```

3.6.16. Множества

Множество — это набор уникальных элементов. В последних версиях некоторых Web-браузеров для создания множества можно воспользоваться классом `Set`. Пример использования класса `Set` приведен в листинге 3.25.

Листинг 3.25. Класс `Set`

```
var arr = [ 1, 2, 3, 1, 2, 3 ];
// Останутся только уникальные элементы
var mySet = new Set(arr);
console.log(mySet); // Set [ 1, 2, 3 ]
// Преобразование множества в массив
var arr2 = Array.from(mySet);
console.log(arr2);  // Array [ 1, 2, 3 ]
// Добавление элементов
mySet.add(4);
mySet.add(2); // Не будет добавлен, т. к. значение не уникально
// Получение числа элементов
console.log(mySet.size);      // 4
// Проверка наличия элемента
console.log(mySet.has(1));    // true
console.log(mySet.has(5));    // false
// Получение списка ключей и значений
console.log([...mySet.keys()]);   // Array [ 1, 2, 3, 4 ]
console.log([...mySet.values()]); // Array [ 1, 2, 3, 4 ]
// Перебор элементов
for (var value of mySet) {
   console.log(value);
}
mySet.forEach(function(value) {
   console.log(value);
});
// Удаление элементов
console.log(mySet.delete(4)); // true
console.log(mySet.size);      // 3
mySet.clear();
console.log(mySet.size);      // 0
```

3.7. Строки

Строки являются упорядоченными последовательностями символов. Следует учитывать, что символы внутри строки изменить нельзя, поэтому все строковые методы в качестве значения возвращают новую строку.

3.7.1. Инициализация строк

В языке JavaScript для хранения строк предназначен тип данных `string`:

```
var str = "Строка";
console.log(typeof str); // string
```

Строка может быть указана как внутри двойных кавычек, так и внутри апострофов (разницы между этими способами нет):

```
var str = 'Строка';
console.log(str); // Строка
```

Создать строку можно также с помощью класса `String` по следующей схеме:

```
<Экземпляр класса> = new String(<Значение>);
```

Пример:

```
var str = new String('Строка');
console.log(typeof str); // object
```

При создании экземпляра класса `String` тип данных будет `object`, а не `string`. Чтобы получить обычную строку, нужно воспользоваться методом `toString()` или `valueOf()`:

```
var str = new String("Строка");
var str2 = str.toString();
console.log(str2);                  // Строка
console.log(typeof str2);           // string
console.log(typeof str.valueOf()); // string
```

Таким способом создания строки пользуются редко. Все методы класса `String` доступны и при использовании элементарного типа. Преобразование элементарного типа в объектный выполняется автоматически:

```
var str = "Строка";
var str2 = str.toUpperCase(); // Перевод символов в верхний регистр
console.log(str2);             // СТРОКА
console.log(typeof str2);      // string
```

При использовании метода `toUpperCase()` строка, имеющая тип данных `string`, автоматически преобразуется в экземпляр класса `String`. Затем производится изменение (в нашем случае — перевод символов в верхний регистр) и возвращается строка, имеющая тип данных `string`. Таким образом, класс `String` является классом-оберткой над элементарным типом `string`.

Для преобразования значения в строку можно использовать следующий формат:

```
<Переменная> = String(<Значение>);
```

Пример:

```
var str = String(10);
console.log(str);          // 10
console.log(typeof str);   // string
```

Статический метод `fromCharCode(<Код 1>,...,<Код N>)` из класса `String` позволяет создать строку из указанных кодов символов в кодировке Unicode:

```
var str = String.fromCharCode(1055, 1088, 1080, 1074, 1077, 1090);
console.log(str);           // Привет
console.log(typeof str);    // string
```

3.7.2. Специальные символы в строке

Специальные символы — это комбинации знаков, обозначающие служебные или непечатаемые символы, которые невозможно вставить обычным способом. Приведем специальные символы, доступные в языке JavaScript:

- `\n` — перевод строки;
- `\r` — возврат каретки;
- `\f` — перевод страницы;
- `\t` — знак табуляции;
- `\v` — знак вертикальной табуляции;
- `\'` — апостроф;
- `\"` — кавычка;
- `\\` — обратная косая черта.

Именно с помощью специального символа `\n` (перевод строки) мы можем разбить сообщение в диалоговом окне на несколько строк:

```
alert("Строка1\nСтрока2\n\nСтрока4");
```

С помощью последовательности `\uNNNN` можно вставить в строку символ в кодировке Unicode:

```
console.log("\u005B");     // [
console.log("\u005D");     // ]
```

Кодировка Unicode содержит не только буквы и цифры, но и различные символы, которые можно использовать на страницах как значки вместо изображений. Выведем символ в виде ножниц:

```
console.log("\u2702");     // Символ "ножницы"
```

3.7.3. Конкатенация строк

Для объединения строк (*конкатенации*) предназначен оператор `+`. Для чисел этот оператор выполняет сложение, но если с одной из сторон от оператора `+` находится строка, то второй операнд преобразуется в строку, а не в число:

```
var str = "Строка1" + "Строка2";
console.log( str ); // Строка1Строка2
```

Часто необходимо сформировать строку, состоящую из имени переменной и ее значения. Если написать так:

```
var x = 10;
var str = "Значение равно x";
console.log( str ); // Значение равно x
```

то переменная `str` будет содержать значение "`Значение равно x`", а если так:

```
var x = 10;
var str = "Значение равно " + x;
console.log( str ); // Значение равно 10
```

то переменная `str` будет содержать значение "`Значение равно 10`".

Для объединения строк можно также воспользоваться оператором `+=`:

```
var str = "Строка1";
str += "Строка2";
console.log( str ); // Строка1Строка2
```

3.7.4. Определение длины строки

Получить длину строки в символах позволяет свойство `length`:

```
var str = "Строка";
console.log( str.length ); // 6
```

3.7.5. Обращение к отдельному символу в строке

К символу строки можно обратиться как к элементу массива. Достаточно указать его индекс в квадратных скобках. Нумерация начинается с нуля. Можно только получить символ по индексу, а вот изменить символ по индексу нельзя:

```
var str = "Строка";
console.log( str[1] ); // т
```

Вместо квадратных скобок лучше использовать метод `charAt(<Индекс символа>)`:

```
var str = "Строка";
console.log( str.charAt(0) ); // С
console.log( str.charAt(1) ); // т
```

Метод `charCodeAt(<Индекс символа>)` позволяет получить код символа в кодировке Unicode:

```
var str = "Строка";
console.log( str.charCodeAt(0) ); // 1057
console.log( str.charCodeAt(1) ); // 1090
```

Строки поддерживают итерации, поэтому мы можем перебрать все символы в цикле:

```
for (var char of "Строка") {
   console.log(char);
}
```

Можно также преобразовать строку в массив, в котором каждый элемент будет содержать только одну букву:

```
var arr = Array.from("Строка");
console.log(arr); // Array [ "С", "т", "р", "о", "к", "а" ]
```

3.7.6. Изменение регистра символов

Для изменения регистра символов предназначены следующие методы:

- `toLowerCase()` — преобразует символы к нижнему регистру:

  ```
  var str = "строка СТРОКА СтРоКа";
  console.log( str.toLowerCase() ); // строка строка строка
  ```

- `toUpperCase()` — преобразует символы к верхнему регистру:

  ```
  var str = "строка СТРОКА СтРоКа";
  console.log( str.toUpperCase() ); // СТРОКА СТРОКА СТРОКА
  ```

- `toLocaleLowerCase()` — преобразует символы к нижнему регистру, учитывая настройки локали:

  ```
  var str = "строка СТРОКА СтРоКа";
  console.log( str.toLocaleLowerCase() ); // строка строка строка
  ```

- `toLocaleUpperCase()` — преобразует символы к верхнему регистру, учитывая настройки локали:

  ```
  var str = "строка СТРОКА СтРоКа";
  console.log( str.toLocaleUpperCase() ); // СТРОКА СТРОКА СТРОКА
  ```

3.7.7. Получение фрагмента строки

Получить фрагмент строки позволяют следующие методы:

- `substr(<Начало фрагмента>[, <Длина фрагмента>])` — извлекает фрагмент строки заданной длины. Если второй параметр пропущен, возвращаются все символы до конца строки:

  ```
  var str = "строка";
  console.log(str.substr(0, 1)); // с
  console.log(str.substr(1));    // трока
  ```

- `substring(<Начало фрагмента>[, <Конец фрагмента>])` — также извлекает фрагмент строки, заданный в этом случае индексами начального и конечного символов. Последний символ во фрагмент не включается:

  ```
  var str = "строка";
  console.log(str.substring(0, 1)); // с
  console.log(str.substring(1, 4)); // тро
  console.log(str.substring(1));    // трока
  ```

- `slice(<Начало фрагмента>[, <Конец фрагмента>])` — также извлекает фрагмент строки, заданный в этом случае индексами начального и конечного символов. Последний символ во фрагмент не включается:

```
var str = "строка";
console.log(str.slice(0, 1));     // с
console.log(str.slice(1, 4));     // тро
console.log(str.slice(1));        // трока
```

Различие между методами `substring()` и `slice()` проявляется при использовании отрицательных индексов. Метод `substring()` трактует отрицательное значение как `0`, а метод `slice()` вычитает это значение из длины строки:

```
var str = "строка";
console.log(str.substring(-2));      // строка
console.log(str.slice(-2));          // ка
console.log(str.substring(1, -2));   // с
console.log(str.slice(1, -2));       // тро
```

3.7.8. Сравнение строк

Для сравнения строк можно использовать операторы сравнения:

```
console.log("строка" == "строка");   // true
console.log("строка1" > "строка2");  // false
console.log("строка1" < "строка2");  // true
```

Сравнение зависит от регистра символов. Кроме того, следует учитывать, что буква «ё» не входит в диапазон от «а» до «я»:

```
console.log("я" > "ё");            // false
console.log("я".charCodeAt(0));   // 1103
console.log("ё".charCodeAt(0));   // 1105
console.log("е".charCodeAt(0));   // 1077
console.log("ж".charCodeAt(0));   // 1078
```

Для сравнения строк с учетом настроек локали следует воспользоваться методом `str1.localeCompare(str2)`. Метод возвращает:

- `положительное число` — если `str1` больше `str2`;
- `отрицательное число` — если `str1` меньше `str2`;
- `0` — если строки равны.

Пример:

```
console.log("я".localeCompare("ё"));  // 1
console.log("ё".localeCompare("я"));  // -1
console.log("я".localeCompare("я"));  // 0
```

3.7.9. Поиск и замена в строке

Для поиска и замены в строке используются следующие методы:

- `indexOf(<Подстрока>[, <Начальная позиция поиска>])` — возвращает индекс позиции первого вхождения подстроки в текущей строке. Если второй параметр не

задан, то поиск начинается с начала строки. Если подстрока не найдена, возвращается значение -1:

```
var str = "строка строка";
console.log(str.indexOf("стр"));        // 0
console.log(str.indexOf("стр", 3));     // 7
console.log(str.indexOf("Стр"));        // -1
```

- `lastIndexOf(<Подстрока>[, <Начальная позиция поиска>])` — определяет индекс позиции последнего вхождения подстроки в текущей строке. Поиск ведется от конца к началу. Если подстрока не найдена, возвращается значение -1:

```
var str = "строка строка";
console.log(str.lastIndexOf("стр"));    // 7
console.log(str.lastIndexOf("стр", 6)); // 0
console.log(str.lastIndexOf("Стр"));    // -1
```

- `startsWith(<Подстрока>[, <Начальная позиция поиска>])` — возвращает значение `true`, если строка начинается с подстроки, и `false` — в противном случае:

```
var str = "строка";
console.log(str.startsWith("стр"));    // true
console.log(str.startsWith("ока", 3)); // true
console.log(str.startsWith("Стр"));    // false
```

- `endsWith(<Подстрока>[, <Длина>])` — возвращает значение `true`, если строка заканчивается подстрокой, и `false` — в противном случае:

```
var str = "строка";
console.log(str.endsWith("ока"));      // true
console.log(str.endsWith("стр", 3));   // true
console.log(str.endsWith("окА"));      // false
```

- `trim()` — удаляет пробельные символы в начале и конце строки:

```
var str = " строка \n\t\v\r";
console.log("'" + str.trim() + "'");   // 'строка'
```

- `trimLeft()` — удаляет пробельные символы в начале строки (метод не входит в стандарт языка и может не работать в некоторых Web-браузерах):

```
var str = "   строка    ";
console.log("'" + str.trimLeft() + "'");  // 'строка    '
```

- `trimRight()` — удаляет пробельные символы в конце строки (метод не входит в стандарт языка и может не работать в некоторых Web-браузерах):

```
var str = "   строка    ";
console.log("'" + str.trimRight() + "'"); // '   строка'
```

- `search(<Регулярное выражение>)` — определяет индекс позиции первого вхождения подстроки, совпадающей с регулярным выражением;
- `match(<Регулярное выражение>)` — возвращает массив с результатами поиска, совпадающими с регулярным выражением;

- `replace(<Регулярное выражение>, <Текст для замены>)` — возвращает строку, которая является результатом поиска и замены в исходной строке с использованием регулярного выражения.

Примеры использования последних трех методов мы рассмотрим при изучении регулярных выражений и встроенного класса `RegExp` (см. *разд. 3.8*).

3.7.10. Преобразование строки в массив

Метод `split(<Разделитель>[, <Лимит>])` возвращает массив, полученный в результате разделения строки на подстроки по разделителю. Если второй параметр присутствует, то он задает максимальное количество элементов в результирующем массиве:

```
var str = "A\tB\tC\tD";
var arr = str.split("\t");
console.log(arr);            // Array [ "A", "B", "C", "D" ]
arr = str.split("\t", 3);
console.log(arr);            // Array [ "A", "B", "C" ]
```

Если в первом параметре указать пустую строку, то каждый элемент массива будет содержать по одной букве:

```
var str = "строка";
var arr = str.split("");
console.log(arr);            // Array [ "с", "т", "р", "о", "к", "а" ]
```

3.7.11. URL-кодирование строк

Выполнить URL-кодирование строки позволяют следующие глобальные функции:

- `encodeURI(<URL-адрес>)` — кодирует URL-адрес целиком;
- `decodeURI(<Строка>)` — декодирует строку, закодированную функцией `encodeURI()`:

  ```
  var url = "test.php?id=5&n=тест";
  var str = encodeURI(url);
  console.log(str); // test.php?id=5&n=%D1%82%D0%B5%D1%81%D1%82
  console.log(decodeURI(str)); // test.php?id=5&n=тест
  ```

- `encodeURIComponent(<Строка>)` — выполняет URL-кодирование строки:

  ```
  var str = encodeURIComponent("Строка");
  console.log(str); // %D0%A1%D1%82%D1%80%D0%BE%D0%BA%D0%B0
  console.log(decodeURIComponent(str)); // Строка
  ```

  В отличие от функции `encodeURI()`, заменяет все спецсимволы шестнадцатеричными кодами:

  ```
  var url = "test.php?n=тест";
  var str = encodeURIComponent(url);
  console.log(str); // test.php%3Fn%3D%D1%82%D0%B5%D1%81%D1%82
  console.log(decodeURIComponent(str)); // test.php?n=тест
  ```

- decodeURIComponent(<Строка>) — декодирует строку, закодированную функцией encodeURIComponent().

3.7.12. Выполнение команд, содержащихся в строке

Глобальная функция eval(<Строка>) выполняет выражение JavaScript, хранящееся в строке:

```
var str = "3 + 5";
var x = eval(str);
console.log(x);         // 8
console.log(typeof x); // number
```

3.8. Регулярные выражения

Регулярные выражения предназначены для сложного поиска или замены в строке. Использовать регулярные выражения в языке JavaScript позволяет класс RegExp.

3.8.1. Создание шаблона

Создать шаблон регулярного выражения можно двумя способами:

```
<Шаблон> = new RegExp(<Регулярное выражение>[, <Модификатор>]);
<Шаблон> = /<Регулярное выражение>/[<Модификатор>];
```

Необязательный параметр <Модификатор> задает дополнительные параметры поиска. Он может содержать следующие символы (или их комбинацию):

- i — поиск без учета регистра;
- g — глобальный поиск (поиск всех вхождений регулярного выражения в строке);
- m — многострочный режим. Символ ^ соответствует началу каждой подстроки, а символ $ — концу каждой подстроки:

```
var p = new RegExp("^[0-9]$", "mg");
var str = "1\n2\n3\nстрока\n4";
var arr = str.match(p);
console.log(arr); // Array [ "1", "2", "3", "4" ]
```

3.8.2. Методы класса *String*

При изучении класса String нами были оставлены без внимания три метода: search(), match() и replace(). Рассмотрим их подробно:

- search(<Регулярное выражение>) — возвращает индекс позиции первого вхождения подстроки, совпадающей с регулярным выражением, или значение -1, если совпадений нет:

```
var p = /20[14]/;
var str = "200, 201, 202, 203, 204";
```

```
console.log(str.search(p));   // 5
console.log("200".search(p)); // -1
```

Шаблону `20[14]` соответствуют только два числа: `201` и `204`;

- `match(<Регулярное выражение>)` — возвращает массив с результатами поиска, совпадающими с регулярным выражением, или значение `null`, если совпадений нет:

```
var p = /20[14]/;
var str = "200, 201, 202, 203, 204";
var arr = str.match(p);
console.log(arr);            // Array [ "201" ]
console.log("200".match(p)); // null
```

Этот пример выведет только `201`, т. к. не указан модификатор глобального поиска `g`. Модифицируем шаблон, чтобы получить все вхождения:

```
var p = /20[14]/g;
var str = "200, 201, 202, 203, 204";
var arr = str.match(p);
console.log(arr);            // Array [ "201", "204" ]
```

Теперь будут выведены все подстроки, совпадающие с регулярным выражением;

- `replace(<Регулярное выражение>, <Текст для замены>)` — возвращает строку, которая является результатом поиска и замены в исходной строке с использованием регулярного выражения:

```
var p = /20[14]/g;
var str = "200, 201, 202, 203, 204";
var str2 = str.replace(p, "207");
console.log(str2);          // 200, 207, 202, 203, 207
```

В качестве первого параметра можно указать строку, но в этом случае будет произведена замена только первого вхождения подстроки в исходную строку:

```
var str = "200, 201, 200, 200";
var str2 = str.replace("200", "207");
console.log(str2);          // 207, 201, 200, 200
```

В качестве второго параметра можно указать ссылку на функцию. Через первый параметр в функции доступна подстрока, полностью соответствующая шаблону. Через следующие параметры доступны подвыражения, которые соответствуют фрагментам, заключенным в шаблоне в круглые скобки. Через предпоследний параметр — смещение подстроки внутри строки, а через последний — исходная строка. Функция должна вернуть подстроку для замены.

В качестве примера найдем все числа в строке и прибавим к ним число `10`:

```
var p = /([0-9]+)/g;
var str = "200, 201, 202, 203, 204";
```

```
var str2 = str.replace(p, function(s, x, offset, source_str) {
   console.log(x + " " + offset + " " + source_str);
   var n = parseInt(s, 10);
   n += 10;
   return n + "";
});
console.log(str2); // 210, 211, 212, 213, 214
```

В строке для замены можно использовать специальные переменные `$1`, ..., `$N`, через которые доступны фрагменты, заключенные в шаблоне в круглые скобки. Через переменную `$&` доступна вся подстрока, через переменную `` $` `` — фрагмент до подстроки, а через переменную `$'` — фрагмент после подстроки.

В качестве примера поменяем два тега местами:

```
var p = /<([a-z]+)><([a-z]+)>/g;
var str = "<br><hr>";
var str2 = str.replace(p, "<$2><$1>");
console.log(str2); // <hr><br>
```

Метод `split(<Регулярное выражение>[, <Лимит>])` также поддерживает регулярные выражения. Он возвращает массив, полученный в результате разделения строки на подстроки по фрагменту, соответствующему регулярному выражению. Если второй параметр присутствует, то он задает максимальное количество элементов в результирующем массиве:

```
var p = /\s/g;
var str = "1 2 3\n4\t5\r6";
var arr = str.split(p);
console.log(arr);          // Array [ "1", "2", "3", "4", "5", "6" ]
arr = str.split(p, 3);
console.log(arr);          // Array [ "1", "2", "3" ]
```

3.8.3. Методы класса *RegExp*

Вместо методов класса `String` можно воспользоваться методами класса `RegExp`:

- `test(<Строка>)` — возвращает `true` или `false` в зависимости от того, был поиск успешным или нет. Если используется глобальный поиск, то при каждом вызове метода указатель текущей позиции будет перемещаться в конец текущего сопоставления:

```
var p = /20[14]/g;
var str = "200, 201, 202, 203, 204";
console.log(p.test(str));    // true
console.log(p.test(str));    // true
console.log(p.test(str));    // false
console.log(p.test("200"));  // false
```

- `exec(<Строка>)` — позволяет получить массив с результатами поиска, совпадающими с регулярным выражением. Если используется глобальный поиск, то при каждом вызове метода указатель текущей позиции будет перемещаться в конец текущего сопоставления. Если совпадений нет, то метод вернет значение `null`:

  ```
  var p = /([0-9]{2}):([0-9]{2}):([0-9]{2})/g;
  var str = "18:47:27 05:12:22";
  console.log(p.exec(str));
  // Array [ "18:47:27", "18", "47", "27" ]
  console.log(p.exec(str));
  // Array [ "05:12:22", "05", "12", "22" ]
  console.log(p.exec(str));     // null
  console.log(p.exec("200"));  // null
  ```

  Первый элемент массива соответствует найденному фрагменту. Второй, третий и четвертый элементы содержат фрагменты, соответствующие группам метасимволов `([0-9]{2})`, заключенным в круглые скобки. Номер скобок по порядку следования в регулярном выражении соответствует индексу фрагмента в массиве.

Если нужно проверить, чтобы строка полностью соответствовала шаблону, следует в шаблоне указать привязку к началу (`^`) и концу (`$`) строки. В качестве примера проверим правильность введенной даты (листинг 3.26).

Листинг 3.26. Проверка правильности введенной даты

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Проверка вводимых данных</title>
</head>
<body>
<script>
var d = window.prompt("Введите дату в формате день.месяц.год", "");
if (d === null) {
   document.write("Вы нажали Отмена");
}
else {
   var p = /^[0-3][0-9]\.[01][0-9]\.[0-9]{4}$/;
   if (p.test(d)) document.write("Дата введена правильно");
   else document.write("Вы неправильно ввели дату");
}
</script>
</body>
</html>
```

3.8.4. Свойства класса *RegExp*

Объект регулярного выражения содержит следующие свойства:

- `lastIndex` — содержит позицию начала поиска (свойство доступно для чтения и записи):

```
var p = /[0-9]+/g;
var str = "1 2 3";
console.log(p.lastIndex);  // 0
console.log(p.exec(str));  // Array [ "1" ]
console.log(p.lastIndex);  // 1
console.log(p.exec(str));  // Array [ "2" ]
console.log(p.lastIndex);  // 3
```

- `global` — содержит значение `true`, если установлен модификатор `g`, и `false` — в противном случае (свойство доступно только для чтения):

```
var p = /[0-9]+/g;
console.log(p.global);        // true
console.log(/[0-9]+/.global); // false
```

- `multiline` — содержит значение `true`, если установлен модификатор `m`, и `false` — в противном случае (свойство доступно только для чтения);
- `ignoreCase` — содержит значение `true`, если установлен модификатор `i`, и `false` — в противном случае (свойство доступно только для чтения);
- `source` — содержит регулярное выражение в виде строки без модификаторов:

```
var p = /[0-9]+/g;
console.log(p.source);        // [0-9]+
```

3.8.5. Синтаксис регулярных выражений

Обычные символы, не имеющие специального значения, могут присутствовать в шаблоне, и они будут трактоваться как есть. Вот пример указания в шаблоне последовательности обычных символов:

```
var p = /строка/;
console.log(p.test("строка")); // true
```

Метасимволы

Как мы уже видели в приведенных ранее примерах, в регулярных выражениях присутствуют специальные символы — так называемые *метасимволы*. Они не всегда соответствуют отдельным символам строки, а управляют тем, как производится проверка строк. Два метасимвола позволяют осуществить привязку выражения к началу или концу строки:

- `^` — привязка к началу строки. Если указан модификатор `m`, то соответствует началу каждой подстроки;

- `$` — привязка к концу строки. Если указан модификатор `m`, то соответствует концу каждой подстроки.

Рассмотрим на примере, как действует привязка:

```
var p = /^[0-9]+$/; // Строка может содержать только числа
if (p.test("2")) console.log("Число"); // Выведет: Число
else console.log("Не число");
if (p.test("Строка2")) console.log("Число");
else console.log("Не число");          // Выведет: Не число
```

Если убрать привязку к началу и концу строки, то любая строка, содержащая число, вернет "Число":

```
var p = /[0-9]+/;
if (p.test("Строка2"))
     console.log("Есть число");        // Выведет: Есть число
else console.log("Нет числа");
```

Можно указать привязку только к началу или только к концу строки:

```
var p = /[0-9]+$/;
if (p.test("Строка2")) console.log("Есть число в конце строки");
else console.log("Нет числа в конце строки");
// Выведет: Есть число в конце строки
p = /^[0-9]+/;
if (p.test("Строка2")) console.log("Есть число в начале строки");
else console.log("Нет числа в начале строки");
// Выведет: Нет числа в начале строки
```

Квадратные скобки `[]` позволяют указать несколько символов, которые могут встречаться на этом месте в строке. Можно привести символы подряд или указать диапазон через тире:

- `[09]` — соответствует числу `0` или `9`;
- `[0-9]` — соответствует любому числу от `0` до `9`;
- `[абв]` — соответствует буквам `а`, `б` и `в`;
- `[а-г]` — соответствует буквам `а`, `б`, `в` и `г`;
- `[а-яё]` — соответствует любой букве от `а` до `я`;
- `[АБВ]` — соответствует буквам `А`, `Б` и `В`. Обратите внимание: если не указан модификатор `i`, регистр будет иметь значение;
- `[А-ЯЁ]` — соответствует любой букве от `А` до `Я`;
- `[а-яА-ЯёЁ]` — соответствует любой русской букве в любом регистре;
- `[0-9а-яА-ЯёЁa-zA-Z]` — любая цифра и любая русская или английская буква независимо от регистра.

> ***Внимание!***
>
> Буква `ё` не входит в диапазон `[а-я]`.

Значение можно инвертировать, если после первой скобки указать символ ^. Таким способом можно указать символы, которых не должно быть на этом месте в строке:

- [^09] — не цифра 0 или 9;
- [^0-9] — не цифра от 0 до 9;
- [^а-яА-ЯёЁa-zA-Z] — не русская или английская буква в любом регистре.

Метасимвол . (точка) соответствует любому символу, кроме символа перевода строки (\n).

Метасимвол | позволяет задать выбор из двух альтернатив. Так, выражение n|m соответствует одному из символов: n или m:

красн(ая)|(ое) — красная или красное, но не красный.

Стандартные классы

Вместо прямого указания символов можно использовать стандартные классы:

- \d — соответствует любой цифре;
- \w — соответствует любой латинской букве, цифре и знаку подчеркивания;
- \s — любой пробельный символ (пробел, табуляция, перевод страницы, новая строка или перевод каретки);
- \D — не цифра;
- \W — не латинская буква, не цифра и не знак подчеркивания;
- \S — не пробельный символ.

> **Внимание!**
> Метасимвол \w работает только с буквами латинского алфавита. С буквами русского языка он не работает.

Экранирование специальных символов

Что же делать, если нужно найти точку, ведь метасимвол «точка» соответствует любому символу, кроме символа перевода строки? Для этого перед специальным символом необходимо указать символ \ (листинг 3.27).

Листинг 3.27. Проверка правильности введенной даты

```
var str = "29,10.2017";
// Неправильная дата (вместо точки указана запятая)
var p = /^[0-3]\d.[01]\d.[12][09]\d\d$/;
// Символ "\" не указан перед точкой
if (p.test(str)) console.log("Дата введена правильно");
else console.log("Дата введена неправильно");
// Выведет "Дата введена правильно", т. к. точка означает любой символ
p = /^[0-3]\d\.[01]\d\.[12][09]\d\d$/;
```

```
// Символ "\" указан перед точкой
if (p.test(str)) console.log("Дата введена правильно");
else console.log("Дата введена неправильно");
// Выведет "Дата введена неправильно",
// т. к. перед точкой указан символ "\", а в дате присутствует запятая

p = new RegExp("^[0-3]\\d\\.[01]\\d\\.[12][09]\\d\\d$");
// Символ "\" указан перед точкой
if (p.test(str)) console.log("Дата введена правильно");
else console.log("Дата введена неправильно");
// Выведет "Дата введена неправильно",
// т. к. перед точкой указан символ "\"
```

Обратите особое внимание на регулярное выражение в последнем примере:

```
"^[0-3]\\d\\.[01]\\d\\.[12][09]\\d\\d$"
```

В строке символ `\` должен заменяться на `\\`. Поэтому вместо `\d` указано `\\d`, а вместо `\.` — `\\.`. Если этого не сделать, в первом случае Web-браузер сообщит об ошибке, а во втором случае — точка будет соответствовать любому символу, кроме символа перевода строки.

Напомним специальные символы, доступные в JavaScript, которые также доступны в регулярных выражениях:

- `\n` — перевод строки;
- `\r` — возврат каретки;
- `\f` — перевод страницы;
- `\t` — знак табуляции;
- `\v` — знак вертикальной табуляции;
- `\uNNNN` — позволяет вставить символ в кодировке Unicode.

Метасимвол «точка» теряет свое специальное значение, если его заключить в квадратные скобки. Кроме того, внутри квадратных скобок могут встретиться символы, которые имеют специальное значение (например, `^` и `-`). Символ `^` теряет свое специальное значение, если он не расположен сразу после открывающей квадратной скобки:

```
var p = /[09^]/g;   // 0, 9 или ^
```

Чтобы отменить специальное значение символа `-`, его необходимо указать после всех символов перед закрывающей квадратной скобкой:

```
var p = /[09-]/g;   // 0, 9 или -
```

Все специальные символы можно сделать обычными, если перед ними указать символ `\`:

```
var p = /[0\-9]/g;  // 0, - или 9
```

Квантификаторы

Количество вхождений символа в строку задается с помощью *квантификаторов*:

- {n} — n вхождений символа в строку (шаблон "[0-9]{2}" соответствует двум вхождениям любой цифры);
- {n,} — n или более вхождений символа в строку (шаблон "[0-9]{2,}" соответствует двум и более вхождениям любой цифры);
- {n,m} — не менее n и не более m вхождений символа в строку. Числа указываются через запятую без пробела. Например, шаблон "[0-9]{2,4}" соответствует от двух до четырех вхождений любой цифры;
- * — ноль или большее число вхождений символа в строку. Эквивалентно комбинации {0,};
- + — одно или большее число вхождений символа в строку. Эквивалентно комбинации {1,};
- ? — ни одного или одно вхождение символа в строку. Эквивалентно комбинации {0,1}.

«Жадность» квантификаторов

Все квантификаторы являются «жадными» — при поиске соответствия ищется самая длинная подстрока, соответствующая шаблону, и не учитываются более короткие соответствия. Рассмотрим это на примере, для чего получим содержимое всех тегов <b> вместе с тегами:

```
var p = /<b>.*<\/b>/gi;
var str = "<b>Text1</b>Text2<b>Text3</b>";
console.log(str.match(p));
// Array [ "<b>Text1</b>Text2<b>Text3</b>" ]
```

Вместо желаемого результата мы получили полностью строку. Чтобы ограничить «жадность», необходимо после квантификатора указать символ ?:

```
var p = /<b>.*?<\/b>/gi;
var str = "<b>Text1</b>Text2<b>Text3</b>";
console.log(str.match(p));
// Array [ "<b>Text1</b>", "<b>Text3</b>" ]
```

Этот код выведет то, что мы искали.

Группы

Если необходимо получить содержимое без тегов, то нужный фрагмент внутри шаблона следует разместить внутри круглых скобок:

```
var p = /<b>(.*?)<\/b>/gi;
var str = "<b>Text1</b>Text2<b>Text3</b>";
var arr;
```

```
while ( (arr = p.exec(str)) ) {
   console.log(arr);
}
// Array [ "<b>Text1</b>", "Text1" ]
// Array [ "<b>Text3</b>", "Text3" ]
```

Результат будет доступен через элемент массива с индексом, совпадающем с порядковым номером круглых скобок внутри шаблона (нумерация начинается с единицы). Элемент массива с индексом `0` будет содержать фрагмент, полностью совпадающий с шаблоном.

Круглые скобки часто используются для группировки фрагментов внутри шаблона. В этих случаях не требуется, чтобы фрагмент запоминался и был доступен в результатах поиска:

```
var p = /[a-z]+((st)|(xt))/gi;
var str = "test text";
var arr;
while ( (arr = p.exec(str)) ) {
   console.log(arr);
}
// Array [ "test", "st", "st", undefined ]
// Array [ "text", "xt", undefined, "xt" ]
```

Три последних элемента в этом примере являются лишними. Чтобы избежать захвата фрагмента после открывающей круглой скобки, следует разместить символы `?:`.

```
var p = /[a-z]+(?:(?:st)|(?:xt))/gi;
var str = "test text";
var arr;
while ( (arr = p.exec(str)) ) {
   console.log(arr);
}
// Array [ "test" ]
// Array [ "text" ]
```

В качестве примера использования групп разберем E-mail (листинг 3.28) и URL-адрес (листинг 3.29) на составные части.

Листинг 3.28. Разбор E-mail на составные части

```
var p = /^([a-z0-9_.-]+)@((?:[a-z0-9-]+\.)+[a-z]{2,6})$/i;
var str = "user@mail.ru";
var arr = p.exec(str);
if (arr) {
   console.log(arr);
   console.log("Имя ящика - " + arr[1]);
   console.log("Имя сайта - " + arr[2]);
   console.log("Полный E-mail - " + arr[0]);
}
```

```
else {
   console.log("E-mail не соответствует шаблону");
}
```

В итоге получим следующий результат:

```
Array [ "user@mail.ru", "user", "mail.ru" ]
Имя ящика - user
Имя сайта - mail.ru
Полный E-mail - user@mail.ru
```

Листинг 3.29. Разбор URL-адреса на составные части

```
var s = "^(\\w+://)" +
        "((?:[a-z0-9-]+\\.)+[a-z]{2,6})" +
        "([a-z0-9/-]*/)*" +
        "([a-z0-9-]+\\.[a-z]+)$";
var p = new RegExp(s, "i");
var str = "http://www.mysite.ru/folder1/folder2/forder3/file.html";
var arr = p.exec(str);
if (arr) {
   console.log("Протокол - " + arr[1]);
   console.log("Сайт - " + arr[2]);
   console.log("Путь - " + arr[3]);
   console.log("Имя файла - " + arr[4]);
}
else {
   console.log("URL не соответствует шаблону");
}
```

В итоге получим следующий результат:

```
Протокол - http://
Сайт - www.mysite.ru
Путь - /folder1/folder2/forder3/
Имя файла - file.html
```

Для получения фрагментов можно также воспользоваться статическими свойствами `$1 ... $N`, а для получения полного соответствия — свойством `lastMatch`. Эти свойства не входят в стандарт, но доступны во всех Web-браузерах:

```
console.log("Полный URL - " + RegExp.lastMatch);
console.log("Протокол - " + RegExp.$1);
console.log("Сайт - " + RegExp.$2);
console.log("Путь - " + RegExp.$3);
console.log("Имя файла - " + RegExp.$4);
```

Обратные ссылки

К найденному фрагменту в круглых скобках внутри шаблона можно обратиться с помощью механизма *обратных ссылок*. Для этого порядковый номер круглых скобок в шаблоне указывается после слеша, например `\1`. Нумерация скобок внутри шаблона начинается с `1`.

Для примера получим текст между одинаковыми парными тегами:

```
var p = /<([a-z]+)[^>]*?>(.*?)<\/\1>/gi;
var str = "<b>Text1</b>Text2<I>Text3</I>";
var arr;
while ( (arr = p.exec(str)) ) {
   console.log(arr);
}
// Array [ "<b>Text1</b>", "b", "Text1" ]
// Array [ "<I>Text3</I>", "I", "Text3" ]
```

3.9. Работа с датой и временем

Работать с датой и временем позволяет класс `Date`. Экземпляры класса создаются так:

```
<Экземпляр класса> = new Date();
<Экземпляр класса> = new Date(<Число миллисекунд>);
<Экземпляр класса> = new Date(<Год>, <Месяц>[, <День>[, <Часы>[,
                         <Минуты>[, <Секунды>[, <Миллисекунды>]]]]]);
<Экземпляр класса> = new Date(<Строка с датой>);
```

3.9.1. Получение текущей даты и времени

При использовании первого формата возвращается объект с текущей датой и временем:

```
var d = new Date();
console.log(d);                  // Date 2017-12-08T17:54:52.625Z
console.log(d.toString());       // Fri Dec 08 2017 20:54:52 GMT+0300
console.log(d.toLocaleString()); // 08.12.2017, 20:54:52
```

Получить число миллисекунд, прошедшее с 1 января 1970 г., позволяет статический метод `now()`:

```
var t = Date.now();
console.log(t);                  // 1512755778548
```

3.9.2. Указание произвольных значений даты и времени

При использовании второго формата дата будет соответствовать числу миллисекунд, прошедших с 1 января 1970 г.:

```
var d = new Date(1512755778548);
console.log(d.toLocaleString()); // 08.12.2017, 20:56:18
```

А при использовании третьего формата можно задать произвольные значения:

```
var d = new Date(2017, 11, 8, 20, 54, 52);
console.log(d.toLocaleString()); // 08.12.2017, 20:54:52
```

Обратите внимание: значения задаются в виде чисел. Кроме того, в параметре <Месяц> указываются числа от 0 (январь) до 11 (декабрь), а не от 1 до 12. Если указать число 12, то месяц станет январем, а к году будет прибавлена единица.

Получить число миллисекунд на основе произвольных значений даты и времени позволяет статический метод UTC():

```
<Время> = Date.UTC(<Год>, <Месяц>[, <День>[, <Часы>[,
                   <Минуты>[, <Секунды>[, <Миллисекунды>]]]]]);
```

Пример:

```
var t = Date.UTC(2017, 11, 8, 20, 54, 52);
console.log(t);                      // 1512766492000
console.log((new Date(t)).toString());
// Fri Dec 08 2017 23:54:52 GMT+0300
```

3.9.3. Разбор строки с датой и временем

Четвертый формат предназначен для разбора строки с датой и временем:

```
var d = new Date("Fri Dec 08 2017 20:54:52 GMT+0300");
console.log(d.toLocaleString()); // 08.12.2017, 20:54:52
```

Получить число миллисекунд, прошедшее с 1 января 1970 г., на основе строки с датой и временем, позволяет статический метод parse():

```
var t = Date.parse("Fri Dec 08 2017 20:54:52 GMT+0300");
console.log(t);                      // 1512755692000
console.log((new Date(t)).toString());
// Fri Dec 08 2017 20:54:52 GMT+0300
```

Если разобрать строку не удалось, то метод parse() вернет значение NaN.

3.9.4. Методы класса *Date*

Класс Date поддерживает следующие основные методы:

- toString() — преобразует дату в строку и возвращает ее:

  ```
  var d = new Date(1512755692000);
  console.log(d.toString());
  // Fri Dec 08 2017 20:54:52 GMT+0300
  ```

- toLocaleString() — преобразует дату в строку, используя настройки локали:

  ```
  var d = new Date(1512755692000);
  console.log(d.toLocaleString()); // 08.12.2017, 20:54:52
  ```

- `toUTCString()` — преобразует дату в строку, используя часовой пояс `UTC`:

```
var d = new Date(1512755692000);
console.log(d.toUTCString());
// Fri, 08 Dec 2017 17:54:52 GMT
```

- `toISOString()` — преобразует дату в строку в формате `ISO`:

```
var d = new Date(1512755692000);
console.log(d.toISOString()); // 2017-12-08T17:54:52.000Z
```

- `valueOf()` и `getTime()` — позволяют определить число миллисекунд, прошедшее с 1 января 1970 г.:

```
var d = new Date(1512755692000);
console.log(d.valueOf()); // 1512755692000
console.log(d.getTime()); // 1512755692000
```

- `getDate()` и `getUTCDate()` — возвращают день месяца (от `1` до `31`):

```
var d = new Date(1512755692000);
console.log(d.getDate());    // 8
console.log(d.getUTCDate()); // 8
```

- `getMonth()` и `getUTCMonth()` — возвращают месяц (от `0` — для января до `11` — для декабря):

```
var months = [ "январь", "февраль", "март", "апрель", "май",
               "июнь", "июль", "август", "сентябрь", "октябрь",
               "ноябрь", "декабрь" ];
var d = new Date(1512755692000);
console.log(d.getMonth());          // 11
console.log(d.getUTCMonth());       // 11
console.log(months[d.getMonth()]); // декабрь
```

Для получения номера текущего месяца к возвращаемому значению необходимо прибавить единицу:

```
var d = new Date(1512755692000);
console.log(d.getMonth() + 1);      // 12
```

- `getFullYear()` и `getUTCFullYear()` — позволяют определить год:

```
var d = new Date(1512755692000);
console.log(d.getFullYear());    // 2017
console.log(d.getUTCFullYear()); // 2017
```

- `getDay()` и `getUTCDay()` — дают возможность узнать день недели (от `0` — для воскресенья до `6` — для субботы):

```
var days = [ "воскресенье", "понедельник", "вторник",
             "среда", "четверг", "пятница", "суббота" ];
var d = new Date(1512755692000);
console.log(d.getDay());       // 5
console.log(d.getUTCDay());    // 5
console.log(days[d.getDay()]); // пятница
```

- `getHours()` и `getUTCHours()` — возвращают час (от `0` до `23`):

  ```
  var d = new Date(1512755692000);
  console.log(d.getHours());    // 20
  console.log(d.getUTCHours()); // 17
  ```

- `getMinutes()` и `getUTCMinutes()` — позволяют получить минуты (от `0` до `59`):

  ```
  var d = new Date(1512755692000);
  console.log(d.getMinutes());    // 54
  console.log(d.getUTCMinutes()); // 54
  ```

- `getSeconds()` и `getUTCSeconds()` — возвращают секунды (от `0` до `59`):

  ```
  var d = new Date(1512755692000);
  console.log(d.getSeconds());    // 52
  console.log(d.getUTCSeconds()); // 52
  ```

- `getMilliseconds()` и `getUTCMilliseconds()` — возвращают миллисекунды (от `0` до `999`):

  ```
  var d = new Date(1512755692156);
  console.log(d.getMilliseconds());    // 156
  console.log(d.getUTCMilliseconds()); // 156
  ```

- `getTimezoneOffset()` — возвращает смещение зоны местного времени в минутах:

  ```
  var d = new Date(1512755692156);
  console.log(d.getTimezoneOffset());  // -180
  ```

Методы, содержащие в названии фрагмент `UTC`, возвращают значения, используя часовой пояс UTC, а без него — по местному времени.

Чтобы задать значения компонентов даты и времени, нужно заменить в названиях методов префикс `get` префиксом `set`:

```
var d = new Date();
d.setDate(8);
d.setMonth(11);
d.setFullYear(2017);
d.setHours(20);
d.setMinutes(54);
d.setSeconds(52);
d.setMilliseconds(156);
console.log(d.toString()); // Fri Dec 08 2017 20:54:52 GMT+0300
```

3.9.5. Вывод даты и времени в окне Web-браузера

Рассмотрим работу с датой и временем на примере (листинг 3.30).

Листинг 3.30. Вывод текущей даты и времени

```
<!DOCTYPE html>
<html lang="ru">
```

```
<head>
 <meta charset="utf-8">
 <title>Текущая дата и время</title>
<script>
function numToStr(value) {
   value += ""; // Преобразуем число в строку
   if (value.length == 1) return "0" + value;
   else return value;
}
function yearToStr(value) {
   value += ""; // Преобразуем число в строку
   return value.substr(2);
}
</script>
</head>
<body>
<script>
var d = new Date();
var days = [ "воскресенье", "понедельник", "вторник", "среда",
             "четверг", "пятница", "суббота" ];
var months = [ "января", "февраля", "марта", "апреля", "мая",
               "июня", "июля", "августа", "сентября", "октября",
               "ноября", "декабря" ];
var msg = "Сегодня <br>" + days[d.getDay()] + " ";
msg += d.getDate() + " ";
msg += months[d.getMonth()] + " ";
msg += d.getFullYear() + " ";
msg += numToStr(d.getHours()) + ":";
msg += numToStr(d.getMinutes()) + ":";
msg += numToStr(d.getSeconds()) + "<br>";
msg += numToStr(d.getDate()) + ".";
msg += numToStr(d.getMonth() + 1) + ".";
msg += yearToStr(d.getFullYear());
document.write(msg);
</script>
</body>
</html>
```

В окне Web-браузера отобразится надпись (в другое время надпись будет иной, т. к. мы работаем с текущим временем):

```
Сегодня
пятница 8 декабря 2017 20:52:20
08.12.17
```

В примере мы использовали две созданные нами функции:

- `numToStr()` — если параметр состоит из одной цифры, то функция добавляет перед ним `0` и возвращает строку. Если не применить функцию, то дата `05.04.2017` будет выглядеть `5.4.2017`, т. к. методы класса `Date` возвращают число;
- `yearToStr()` — функция возвращает последние две цифры года.

3.9.6. Таймеры. Создание часов на Web-странице

Таймеры позволяют однократно или многократно выполнять указанную функцию через определенный интервал времени. Для управления таймерами используются следующие методы объекта `window`:

- `setTimeout()` — создает таймер, однократно выполняющий указанную функцию или выражение спустя заданный интервал времени:

  ```
  <Идентификатор> = setTimeout(<Функция или выражение>,
                               <Интервал>[, <Параметры>]);
  ```

- `clearTimeout(<Идентификатор>)` — останавливает таймер, установленный методом `setTimeout()`;
- `setInterval()` — создает таймер, многократно выполняющий указанную функцию или выражение через заданный интервал времени.

  ```
  <Идентификатор> = setInterval(<Функция или выражение>,
                                <Интервал>[, <Параметры>]);
  ```

- `clearInterval(<Идентификатор>)` — останавливает таймер, установленный методом `setInterval()`.

Здесь `<Интервал>` — это промежуток времени, по истечении которого выполняется `<Функция или выражение>`. Значение указывается в миллисекундах. В аргументе `<Параметры>` можно указать значения через запятую. Эти значения будут переданы в функцию в качестве параметров.

Приведем пример использования таймеров. В листинге 3.31 созданы часы на Web-странице, отображающие время вплоть до секунды. Добавим также возможность остановки и запуска часов.

Листинг 3.31. Часы на Web-странице

```
<!DOCTYPE html>
<html lang="ru">
<head>
 <meta charset="utf-8">
 <title>Часы на Web-странице</title>
<script>
var timerId;
function numToStr(value) {
   value += ""; // Преобразуем число в строку
   if (value.length == 1) return "0" + value;
   else return value;
}
function startTimer() { // Запускаем таймер
   timerId = setInterval(function() {
      var d = new Date();
      var msg = numToStr(d.getHours()) + ":";
```

```
      msg += numToStr(d.getMinutes()) + ":";
      msg += numToStr(d.getSeconds());
      document.getElementById("div1").innerHTML = msg;
   }, 1000);
   document.getElementById("div_start").style.display = "none";
   document.getElementById("div_stop").style.display = "block";
}
function stopTimer() { // Останавливаем таймер
   clearInterval(timerId);
   document.getElementById("div_start").style.display = "block";
   document.getElementById("div_stop").style.display = "none";
}
</script>
</head>
<body onload="startTimer()">
<div id="div1"></div>
<div id="div_start">
  <input type="button" value="Запустить часы" onclick="startTimer()">
</div>
<div id="div_stop">
  <input type="button" value="Остановить часы" onclick="stopTimer()">
</div>
</body>
</html>
```

3.10. Функции. Разделение программы на фрагменты

Функция — это фрагмент кода JavaScript, который можно вызвать из любого места программы. Создание пользовательских функций позволит уменьшить избыточность программного кода и повысить его структурированность.

3.10.1. Создание функции

Функция описывается с помощью ключевого слова `function` по следующей схеме:

```
function <Имя функции>([<Параметры через запятую>]) {
   <Тело функции>
   [return[ <Возвращаемое значение>];]
}
```

Функция должна иметь уникальное имя. Для имен действуют такие же правила, что и при указании имени переменной.

После имени функции в круглых скобках можно указать один или несколько параметров через запятую. Параметров может вообще не быть — в этом случае указываются только круглые скобки.

Между фигурными скобками располагаются инструкции JavaScript, которые будут исполнены после каждого вызова функции. Точка с запятой после закрывающей фигурной скобки не ставится.

Функция может возвращать значение в место вызова функции. Возвращаемое значение задается с помощью ключевого слова `return`. Если ключевое слово `return` не указано или указано без возвращаемого значения, то функция вернет значение `undefined`.

Пример функции без параметров:

```
function showOK() {
   alert("Сообщение при удачно выполненной операции");
}
```

Пример функции с параметром:

```
function showMessage(msg) {
   alert(msg);
}
```

Пример функции с параметрами, возвращающей сумму двух переменных:

```
function sum(x, y) {
   var z = x + y;
   return z;
}
```

В качестве возвращаемого значения в конструкции `return` можно указывать не только имя переменной, но и выражение:

```
function sum(x, y) {
   return x + y;
}
```

В программе функции можно вызвать следующим образом:

```
showOK(); // Сообщение при удачно выполненной операции
showMessage("Сообщение"); // Сообщение
var n = sum(5, 2);        // Переменной n будет присвоено значение 7
showMessage("n = " + n);  // n = 7
```

Инструкции, указанные после конструкции `return`, никогда не будут выполнены:

```
function sum(x, y) {
   return x + y;
   alert("Сообщение"); // Эта инструкция никогда не будет выполнена
}
```

В окне консоли в этом случае отобразится предупреждающее сообщение: **unreachable code after return statement**.

Имя переменной, передающей значение функции, может не совпадать с именем переменной внутри функции:

```
function sum(x, y) {
   return x + y;
}
var var1 = 5, var2 = 2, var3;
var3 = sum(var1, var2);
```

Ссылку на функцию можно сохранить в какой-либо переменной. Для этого название функции указывается без круглых скобок:

```
function test() {
   alert("Это функция test()");
}
var x;
x = test; // Присваиваем ссылку на функцию
x();      // Вызываем функцию test() через переменную x
console.log(typeof test); // function
```

3.10.2. Расположение функций внутри HTML-документа

Обычно функции принято располагать в разделе `HEAD` HTML-документа (листинг 3.32) или в отдельном файле с расширением js (листинги 3.33 и 3.34). Впрочем, функции могут располагаться и в разделе `BODY`.

Листинг 3.32. Функция расположена в разделе `HEAD`

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Функции</title>
   <script>
   function sum(x, y) {
      return x + y;
   }
   </script>
</head>
<body>
   <script>
   var var1 = 5, var2 = 2, var3;
   var3 = sum(var1, var2);
   document.write("var3 = " + var3);
   </script>
</body>
</html>
```

Листинг 3.33. Функция вынесена в отдельный файл script.js

```
<!DOCTYPE html>
<html lang="ru">
```

```
<head>
   <meta charset="utf-8">
   <title>Функции</title>
   <script type="text/javascript" src="script.js"></script>
</head>
<body>
   <script>
   var var1 = 5, var2 = 2, var3;
   var3 = sum(var1, var2);
   document.write("var3 = " + var3);
   </script>
</body>
</html>
```

Листинг 3.34. Содержимое файла script.js

```
function sum(x, y) {
   return x + y;
}
```

3.10.3. Класс *Function*

Класс `Function` позволяет создавать функцию как экземпляр класса. Делается это таким образом:

```
<Переменная> = new Function(<Параметр 1>[, ... , <Параметр N>],
                            <Тело функции>);
```

Например, функцию суммирования двух чисел

```
function sum(x, y) {
   return x + y;
}
```

можно переписать так (оператор `new` можно не указывать):

```
var sum = new Function("x", "y", "return x + y");
console.log(typeof sum); // function
console.log(sum(5, 7));  // 12
```

Указывать тело функции в виде строки очень неудобно. По этой причине данным способом никто не пользуется.

Каждая функция содержит следующие методы из класса `Function`:

- `call([<Объект>[, <Параметры через запятую>]])` — вызывает функцию в контексте объекта `<Объект>` и передает ей значения, указанные во втором и последующих параметрах.

  Вот пример вызова функции как метода объекта `window`:

  ```
  function sum(x, y) {
     return x + y;
  }
  console.log(sum.call(window, 5, 7)); // 12
  ```

Все имена функций, как и имена глобальных переменных, становятся свойствами глобального объекта `window`. Вызов функции в предыдущем примере аналогичен следующему вызову:

```
console.log(window.sum(5, 7));        // 12
```

Объект, указанный в первом параметре метода `call()`, доступен внутри функции через указатель `this`:

```
function sum(x, y) {
   console.log(this); // Window
   return x + y;
}
```

Если первый параметр не указан или имеет значение `null`, то функция вызывается в глобальном контексте:

```
console.log(sum.call(null, 5, 7));   // 12
```

- `apply([<Объект>[, <Массив с параметрами>]])` — метод аналогичен методу `call()`, но значения для функции указываются в виде массива:

```
console.log(sum.apply(window, [5, 7])); // 12
console.log(sum.apply(null, [5, 7]));   // 12
```

3.10.4. Переменное число параметров в функции

Внутри функции доступна локальная переменная `arguments`, которая позволяет получить доступ ко всем параметрам, переданным функции. Получить доступ к параметру можно, указав его индекс внутри квадратных скобок. Свойство `length` позволяет определить количество параметров, переданных функции:

```
function sum(x, y) {
   console.log(arguments.length);       // 2
   return arguments[0] + arguments[1];
}
console.log(sum(5, 6));                 // 11
```

Какой в этом смысл? Дело в том, что при использовании переменной `arguments` можно передать функции больше параметров, чем первоначально объявлено. Например, можно просуммировать сразу несколько чисел, а не только два (листинг 3.35).

Листинг 3.35. Произвольное количество параметров

```
function sum(x, y) {
   var z = 0;
   for (var i = 0; i < arguments.length; i++) {
      z += arguments[i];
   }
   return z;
}
console.log(sum(5, 6, 7, 20)); // 38
```

В последних версиях некоторых Web-браузеров значения, переданные при вызове в большем количестве, чем указано параметров при объявлении, можно поместить в массив (листинг 3.36). Для этого перед именем параметра указываются три точки. Такой параметр должен быть расположен после всех остальных параметров.

Листинг 3.36. Произвольное количество параметров

```
function sum(x, y, ...arr) {
   var z = x + y;
   for (var i = 0; i < arr.length; i++) {
      z += arr[i];
   }
   return z;
}
console.log(sum(5, 6, 7, 20)); // 38
```

Первые два значения будут присвоены переменным `x` и `y`, а все остальные — добавлены в массив `arr`.

3.10.5. Глобальные и локальные переменные

Глобальные переменные — это переменные, объявленные вне функции. Глобальные переменные видны в любой части программы, включая функции. Все глобальные переменные становятся свойствами глобального объекта `window`.

Локальные переменные — это переменные, объявленные внутри функции. Локальные переменные видны только внутри тела функции. Если имя локальной переменной совпадает с именем глобальной переменной, то все операции внутри функции осуществляются с локальной переменной, а значение глобальной не изменяется. Получить значение глобальной переменной внутри функции при наличии одноименной локальной переменной можно через объект `window`.

Механизм, регулирующий такое поведение, называется *областью видимости переменных*. Он продемонстрирован в листинге 3.37.

Листинг 3.37. Глобальные и локальные переменные

```
<!DOCTYPE html>
<html lang="ru">
<head>
 <meta charset="utf-8">
 <title>Глобальные и локальные переменные</title>
<script>
function test() {
   var var1 = 5, var3 = 1;
   document.write("Локальная переменная var1 = " + var1 + "<br>");
```

```
   document.write("Локальная переменная var3 = " + var3 + "<br>");
   document.write("Глобальная переменная var2 = " + var2 + "<br>");
   document.write("Доступ к одноименной глобальной переменной: ");
   document.write("window.var1 = " + window.var1 + "<br>");
}
</script>
</head>
<body>
<script>
var var1 = 10, var2 = 7;
document.write("Глобальная переменная var1  = " + var1 + "<br>");
test();
document.write("Глобальная переменная var1 осталась = ");
document.write(var1 + "<br>");
document.write("Локальная переменная var3 = " + typeof var3);
document.write(", т. е. не видна вне тела функции");
</script>
</body>
</html>
```

В окне Web-браузера получим следующий результат:

```
Глобальная переменная var1 = 10
Локальная переменная var1 = 5
Локальная переменная var3 = 1
Глобальная переменная var2 = 7
Доступ к одноименной глобальной переменной: window.var1 = 10
Глобальная переменная var1 осталась = 10
Локальная переменная var3 = undefined, т. е. не видна вне тела функции
```

Как видно из листинга 3.36, переменная `var3`, объявленная внутри функции `test()`, не доступна вне функции. Глобальную переменную `var1` не затронуло объявление внутри функции одноименной локальной переменной и ее изменение. А глобальная переменная `var2` видна внутри функции `test()`.

3.10.6. Область видимости блока

Во многих языках программирования переменная, объявленная внутри блока (внутри фигурных скобок), не видна после блока. В языке JavaScript при объявлении переменной с помощью ключевого слова `var` это не так. Если мы объявляем переменную внутри блока, например в первом параметре цикла `for`, то переменная будет видна и после закрывающей фигурной скобки:

```
for (var i = 0; i < 5; i++) {
   console.log(i);
}
console.log("i = " + i); // i = 5
```

Результат будет аналогичным при объявлении переменной внутри блока:

```
var x = 5;
if (x == 5) {
   var y = x + 10;
}
console.log("y = " + y); // y = 15
```

В последних версиях некоторых Web-браузеров мы можем объявить переменные с помощью ключевого слова `let`. В этом случае область видимости переменных будет ограничена блоком, в котором переменная объявлена. Переменная будет также видна и во всех вложенных блоках, а вот вне блока она видна не будет:

```
var x = 5;
if (x == 5) {
   let y = x + 10;
}
console.log("y = " + typeof y); // y = undefined
```

При объявлении переменных в первом параметре цикла `for` область видимости будет ограничена телом цикла:

```
for (let i = 0; i < 5; i++) {
   console.log(i);
}
console.log("i = " + typeof i); // i = undefined
```

В одной инструкции можно объявить сразу несколько переменных через запятую и присвоить им начальные значения:

```
{
   let x, y = 10;
   x = y + 5;
   console.log(x + " " + y);    // 15 10
}
console.log("x = " + typeof x); // x = undefined
console.log("y = " + typeof y); // y = undefined
```

В этом примере мы использовали блок без привязки к какой-либо другой конструкции. Этот блок просто ограничивает область видимости переменной.

Повторное объявление переменной приведет к ошибке:

```
{
   let x = 5;
   let x = 20; // SyntaxError: redeclaration of let x
}
```

Ошибка также возникнет при обращении к переменной до ее объявления:

```
{
   console.log("x = " + x);
   // ReferenceError: can't access lexical
   // declaration `x' before initialization
```

```
   let x = 20;
}
```

При использовании ключевого слова `var` мы получили бы значение `undefined`:

```
console.log("x = " + x); // x = undefined
var x = 20;
```

3.10.7. Способы передачи параметров в функцию

При вызове в функцию передается копия значения переменной. Таким образом, изменение значения внутри функции не затронет значение исходной переменной:

```
function test(x) {
   x = 10;
}
var x = 20;
test(x);
console.log("x = " + x); // x = 20
```

Однако при использовании объектов в переменной сохраняется лишь ссылка на объект, а не сам объект. Эта ссылка передается в качестве значения в функцию. Внутри функции мы не можем присвоить переменной ссылку на другой объект, а вот изменить свойства объекта через эту ссылку можем:

```
function test(arr) {
   arr[0] = 33;
}
var a = [1, 2];
test(a);
console.log(a); // Array [ 33, 2 ]
```

Как видно из примера, массив изменился.

3.10.8. Необязательные параметры

Некоторые параметры можно сделать необязательными. Для создания необязательных параметров используется проверка равенства значению `undefined` или оператор `||` (если первое выражение не может быть преобразовано в `true`, то возвращается значение второго выражения):

```
function test(str) {
   str = str || "Значение по умолчанию";
   return str;
}
console.log( test() );                      // Значение по умолчанию
console.log( test("Значение указано") ); // Значение указано
```

В последних версиях некоторых Web-браузеров значения по умолчанию можно указать при объявлении параметров:

```
function test(str="Значение по умолчанию") {
   return str;
}
console.log( test() );                    // Значение по умолчанию
console.log( test("Значение указано") ); // Значение указано
```

Следует учитывать, что значение по умолчанию вычисляется при каждом вызове функции, поэтому в этом примере каждый раз будет создаваться новый массив:

```
function test(x, arr=[]) {
   arr.push(x);
   return arr;
}
console.log( test(2) ); // Array [ 2 ]
console.log( test(5) ); // Array [ 5 ]
```

3.10.9. Анонимные функции

Функция может вообще не иметь названия. В этом случае ссылку на анонимную функцию сохраняют в переменной:

```
var func = function() { // Присваиваем ссылку на анонимную функцию
   alert("Сообщение");
};
func(); // Вызываем анонимную функцию через переменную func
```

Анонимная функция становится видимой только после описания. Если вызов функции поместить перед описанием, то будет выведено сообщение об ошибке. При использовании обычных функций ошибки не будет.

Ссылку на вложенную функцию можно вернуть в качестве значения в конструкции `return`. Чтобы вызвать вложенную функцию, круглые скобки указываются два раза:

```
var func = function() { // Присваиваем ссылку на анонимную функцию
   return function() {  // Возвращаем ссылку на вложенную функцию
      alert("Это вложенная функция");
   };
};
func()(); // Вызываем вложенную функцию через переменную func
```

Используя анонимные функции, следует учитывать, что при указании внутри функции глобальной переменной будет сохранена ссылка на эту переменную, а не на ее значение:

```
var x = 5;
var func = function() {
   return x; // Сохраняется ссылка, а не значение переменной x !
};
console.log(func()); // 5
x = 10;              // Изменили значение
console.log(func()); // 10, а не 5
```

Анонимные функции захватывают переменные в родительской области видимости (это называется *замыканием*). Если анонимная функция объявлена внутри функции, то после выхода из функции область видимости сохраняется, и мы можем получить доступ к переменным, существовавшим в момент создания анонимной функции:

```
function test() {
   var x = 10, y = 5;
   return function() {
      return x + y;
   };
}
var func = test();
console.log(func()); // 15
```

3.10.10. Стрелочные функции

В последних версиях некоторых Web-браузеров доступны также *стрелочные функции*, которые в других языках называются *лямбда-выражениями*. Объявление стрелочной функции имеет следующие форматы:

```
<Переменная> = <Параметр> => <Возвращаемое значение>;
<Переменная> = ([<Параметр 1>[, ..., <Параметр N>]]) =>
                <Возвращаемое значение>;
<Переменная> = ([<Параметр 1>[, ..., <Параметр N>]]) => {
   <Тело функции>
   [return <Возвращаемое значение>;]
};
```

Стрелочные функции, так же как и анонимные функции, не имеют имени. Ссылка на функцию сохраняется в переменной. Чтобы вызвать функцию, нужно после имени переменной указать круглые скобки и внутри них значения.

Вот пример стрелочной функции с одним параметром:

```
var func = n => n * 2;
console.log(func(5)); // 10
```

Если тело функции состоит из одного выражения, то фигурные скобки можно не указывать. Результат вычисления этого выражения станет возвращаемым значением. Если указаны фигурные скобки, то для возврата значения нужно использовать конструкцию `return`. Предыдущий пример можно записать так:

```
var func = n => {
   return n * 2;
};
console.log(func(5)); // 10
```

Если функция принимает только один параметр, то круглые скобки можно не указывать. В противном случае параметры записываются через запятую внутри круглых скобок:

```
var func = (x, y) => x + y;
console.log(func(5, 7)); // 12
```

Круглые скобки обязательно указываются при отсутствии параметров:

```
var func = () => console.log("Сообщение");
func(); // Сообщение
```

Стрелочные функции захватывают переменные в родительской области видимости. Если стрелочная функция объявлена внутри функции, то после выхода из функции область видимости сохраняется, и мы можем получить доступ к переменным, существовавшим в момент создания стрелочной функции:

```
function test() {
   var x = 10, y = 5;
   return () => x + y;
}
var func = test();
console.log(func()); // 15
```

В отличие от обычных функций, внутри стрелочных функций нет доступа к переменной `arguments`. Чтобы передать неопределенное количество параметров, нужно перед именем параметра вставить оператор «три точки»:

```
var sum = (...arr) => {
   var z = 0;
   for (var i = 0; i < arr.length; i++) {
      z += arr[i];
   }
   return z;
};
console.log(sum(5, 6, 7, 20)); // 38
```

В отличие от анонимных функций, стрелочные функции сохраняют ссылку на родительский указатель `this`:

```
function test() {
   console.log(this);                   // Object {  }
   var func1 = function() {
      console.log(this);                // Window
   };
   var func2 = () => console.log(this); // Object {  }
   func1();
   func2();
}
var obj = {};
test.call(obj);
```

Причем указание контекста в методах `call()` и `apply()` не изменяет значение указателя `this`.

3.10.11. Функции-генераторы

В последних версиях некоторых Web-браузеров доступны *функции-генераторы*. Функцией-генератором называется функция, которая может возвращать одно значение из нескольких значений на каждой итерации. Приостановить выполнение функции и превратить функцию в генератор позволяет ключевое слово `yield`. В отличие от обычных функций, после ключевого слова `function` указывается символ звездочка (`function*`). В качестве примера напишем функцию, которая умножает элементы последовательности на указанное число (листинг 3.38).

Листинг 3.38. Функции-генераторы

```
function* test(x, y) {
   for (let i = 1; i <= x; i++) {
      yield i * y;
   }
}
var obj = test(10, 2);
for (let n of obj) {
   console.log(n);
}
obj = test(3, 2);
console.log(obj.next().value); // 2
console.log(obj.next().value); // 4
console.log(obj.next().value); // 6
console.log(obj.next().value); // undefined
```

После вызова функции-генератора возвращается объект, поддерживающий итерации. На каждом шаге возвращается значение, после чего состояние генератора сохраняется. При следующем вызове функции выполнение возобновляется с места, на котором прервалось. Таким образом можно сгенерировать бесконечное число значений (по одному на каждой итерации), не опасаясь превысить лимит оперативной памяти.

Возвращаемый объект содержит метод `next()`, с помощью которого можно перейти к следующему значению. Объект, возвращаемый методом `next()`, содержит два свойства: `value` (позволяет получить текущее значение) и `done` (содержит значение `true`, если больше нет значений, и `false` — в противном случае):

```
obj = test(3, 2);
console.log(obj.next()); // Object { value: 2, done: false }
console.log(obj.next()); // Object { value: 4, done: false }
console.log(obj.next()); // Object { value: 6, done: false }
console.log(obj.next()); // Object { value: undefined, done: true }
```

3.10.12. Рекурсия. Вычисление факториала

Рекурсия — это возможность функции вызывать саму себя. С одной стороны, это удобно, с другой стороны, если не предусмотреть условие выхода, происходит зацикливание. Типичным применением рекурсии является вычисление факториала числа (листинг 3.39).

Листинг 3.39. Вычисление факториала

```
<!DOCTYPE html>
<html lang="ru">
<head>
 <meta charset="utf-8">
 <title>Вычисление факториала</title>
<script>
function factorial(x) {
   if (x <= 1) return 1;
   else return (x * factorial(x - 1));
}
</script>
</head>
<body>
<script>
var z;
z = window.prompt("Вычисление факториала\nВведите число", "");
if (z === null) {
   document.write("Вы нажали Отмена");
}
else {
   var n = parseInt(z, 10);
   if (isNaN(n)) {
      document.write("Вы ввели не целое число");
   }
   else {
      document.write("Факториал числа " + z + " = ");
      document.write(factorial(n));
   }
}
</script>
</body>
</html>
```

3.11. Классы и объекты

Объектно-ориентированное программирование (*ООП*) — это способ организации программы, позволяющий использовать один и тот же код многократно. Основным «кирпичиком» ООП является *класс*. Класс включает набор переменных (называе-

мых *полями* или *свойствами* класса) и функций для управления этими переменными (называемых *методами*). В совокупности свойства и методы называются *членами* класса. После создания класса его название становится новым типом данных.

Следует заметить, что в JavaScript нет полноценной поддержки объектно-ориентированного программирования — такой, как в языках C++ или Java.

3.11.1. Основные понятия

Для использования методов и свойств класса необходимо создать *экземпляр класса*. Для этого используется оператор `new` — после него указывается имя класса, к которому будет относиться созданный экземпляр. После имени класса, в круглых скобках, можно передавать некоторые параметры, задавая таким образом начальные значения свойствам класса:

```
<Экземпляр класса> = new <Имя класса>([<Параметры>]);
```

Пример:

```
var x = new Number(10);
console.log(typeof x);      // object
```

При создании экземпляра класса ссылка (указатель) сохраняется в переменной. Используя ссылку, можно обращаться к свойствам и методам созданного экземпляра класса.

При обращении к свойствам используется следующий формат:

```
<Экземпляр класса>.<Имя свойства>;
```

Обращение к методам осуществляется аналогично, только после имени метода необходимо указать круглые скобки:

```
<Экземпляр класса>.<Имя метода>([<Параметры>]);
```

В скобках часто указываются параметры метода:

```
console.log(x.toString(10));   // 10
```

Для обращения к свойствам и методам можно использовать квадратные скобки, внутри которых указывается имя в виде строки:

```
console.log(x["toString"](10));   // 10
```

3.11.2. Класс *Object*

Создать новый объект можно с помощью встроенного класса `Object` (листинг 3.40).

Листинг 3.40. Класс `Object`

```
var car = new Object();
car.model = "ВАЗ-2109";      // Сохранили строку
car.year = 2007;             // Сохранили число
car.getModel = function() { // Сохранили ссылку на функцию
```

```
   return this.model;
};
// Вывод значений
console.log(car.model);      // ВАЗ-2109
console.log(car.year);       // 2007
console.log(car.getModel()); // ВАЗ-2109
```

После создания объекта в переменной `car` сохраняется ссылка на него. Используя точечную нотацию, можно добавить свойство (переменную внутри объекта). В качестве значения свойства может быть указан любой тип данных: число, строка, массив или другой объект. Если в качестве значения указать ссылку на функцию, то такое свойство становится методом объекта, внутри которого доступен указатель (`this`) на текущий объект.

Класс `Object` содержит метод `toString()`, который наследуется всеми классами. Он возвращает значение объекта в виде строки. Метод вызывается автоматически при использовании объекта в строковом контексте:

```
console.log(car.toString()); // [object Object]
console.log(car + "");       // [object Object]
```

Объектная переменная может содержать специальное значение `null`, которое означает, что переменная не содержит объекта. Это значение удобно возвращать из функции при невозможности создать объект — например, в случае ошибки.

3.11.3. Создание объекта с помощью фигурных скобок

Создать объект можно также с помощью фигурных скобок (листинг 3.41).

Листинг 3.41. Создание объекта с помощью фигурных скобок

```
var car = {
   model: "ВАЗ-2109",
   year: 2007,
   getModel: function() {
      return this.model;
   }
};
// Вывод значений
console.log(car.model);      // ВАЗ-2109
console.log(car.year);       // 2007
console.log(car.getModel()); // ВАЗ-2109
```

В этом случае значение свойства указывается после двоеточия, а пары «свойство/значение» записываются через запятую.

В последних версиях некоторых Web-браузеров создать метод внутри фигурных скобок можно так:

```
var car = {
   model: "ВАЗ-2109",
   year: 2007,
   getModel() {
      return this.model;
   }
};
```

Если между фигурными скобками нет никаких инструкций, то создается пустой объект:

```
var obj = {};                // Пустой объект
console.log(obj);            // Object {  }
```

При создании объектов следует учитывать один очень важный момент. Например, нам необходимо определить два пустых объекта, которые в дальнейшем будут использоваться раздельно. Очень силен соблазн написать следующим образом:

```
var obj1 = obj2 = {};        // Якобы определили два объекта
```

Проблема заключается в том, что в этом примере создается только один объект, а ссылка на него сохраняется в двух переменных. Таким образом, все изменения `obj1` будут отражаться и на переменной `obj2`:

```
var obj1 = obj2 = {};        // Якобы определили два объекта
obj1.test = "Это значение свойства test объекта obj1";
console.log(obj2.test);
// Выведет: Это значение свойства test объекта obj1
```

Помните, что присваивание и сравнение объектов производится по ссылке, а не по значению. Поэтому создавать объекты необходимо раздельно:

```
var obj1 = {}, obj2 = {};
obj1.test = "Это значение свойства test объекта obj1";
console.log(obj2.test); // Выведет: undefined
```

3.11.4. Конструктор класса

Если после ключевого слова `new` указана функция, то она становится *конструктором* класса, которому можно передать начальные данные при инициализации (листинг 3.42). Внутри конструктора доступен указатель (`this`) на текущий объект.

Листинг 3.42. Конструктор класса

```
function Car(m, y) { // Конструктор класса
   this.model = m;
   this.year = y;
   this.getModel = function() {
      return this.model;
   }
}
```

```
// Создание экземпляра
var obj1 = new Car("ВАЗ-2109", 2007);
var obj2 = new Car("LADA Granta", 2014);
// Вывод значений
console.log(obj1.year);       // 2007
console.log(obj1.getModel()); // ВАЗ-2109
console.log(obj2.year);       // 2014
console.log(obj2.getModel()); // LADA Granta
```

С помощью конструктора класса можно создать множество объектов (экземпляров класса). При этом для каждого объекта создается свой набор локальных переменных. Класс можно сравнить с чертежом, а объект — с изделием, произведенным по этому чертежу. Чертеж всегда один, а вот изделий может быть бесконечное множество. Причем каждое изделие уникально и не зависит от других изделий.

3.11.5. Инструкция *class*

В последних версиях некоторых Web-браузеров создать класс можно с помощью ключевого слова `class`:

```
class <Имя класса>[ extends <Базовый класс>] {
   // <Свойства и методы класса>
}
<Переменная> = class[ extends <Базовый класс>] {
   // <Свойства и методы класса>
}
```

Конструктор класса внутри инструкции `class` определяется с помощью метода со специальным названием `constructor()`. Пример создания класса приведен в листинге 3.43.

Листинг 3.43. Инструкция `class`

```
class Car {
   constructor(m, y) { // Конструктор класса
      this.model = m;
      this.year = y;
   }
   getModel() {
      return this.model;
   }
}
// Создание экземпляра
var obj1 = new Car("ВАЗ-2109", 2007);
var obj2 = new Car("LADA Granta", 2014);
// Вывод значений
console.log(obj1.year);       // 2007
```

```
console.log(obj1.getModel()); // ВАЗ-2109
console.log(obj2.year);       // 2014
console.log(obj2.getModel()); // LADA Granta
```

Ключевое слово `extends` позволяет создать цепочку наследования. *Наследование* — это возможность создания производных классов на основе базового класса. При этом производный класс автоматически получает возможности базового класса и может добавить новую функциональность или переопределить некоторые методы. Пример создания иерархии классов приведен в листинге 3.44.

Листинг 3.44. Наследование

```
class A {
   constructor() {  // Конструктор
      console.log("Конструктор класса A");
   }
   func1() {
      console.log("Метод finc1() из класса A");
   }
}
class B extends A {
   constructor() {  // Конструктор
      super();      // Вызов конструктора базового класса
      console.log("Конструктор класса B");
   }
   func1() {
      super.func1(); // Вызов метода базового класса
      console.log("Метод finc1() из класса B");
   }
   func2() {
      console.log("Метод finc2() из класса B");
   }
}
var objB = new B();
// Конструктор класса A
// Конструктор класса B
objB.func2(); // Метод finc2() из класса B
objB.func1();
// Метод finc1() из класса A
// Метод finc1() из класса B
```

В этом примере класс `B` наследует все члены класса `A`. Класс `A` называется базовым классом или суперклассом, а класс `B` — производным классом или подклассом.

Конструктор в базовом классе автоматически не вызывается, если он был переопределен в производном классе. Для передачи значений конструктору базового класса используется следующий синтаксис:

```
super([<Значения>]);
```

Чтобы переопределить метод базового класса, достаточно в производном классе создать одноименный метод. В этом случае будет вызван метод только из производного класса. Чтобы вызвать метод из базового класса, внутри производного класса используется следующий синтаксис:

```
super.<Имя метода>([<Значения>]);
```

3.11.6. Свойства и методы класса

Все рассмотренные ранее варианты позволяли создавать свойства и методы экземпляра класса. Тем не менее можно также создать свойства и методы, связанные с самим классом, а не с его экземпляром (листинг 3.45).

Листинг 3.45. Свойства и методы класса

```
function Car(m) { this.model = m; }
Car.text = "Модель ";
Car.showInfo = function(obj) {
   console.log(Car.text + obj.model);
};
var obj = new Car("ВАЗ-2109");
// Вывод значений
console.log(Car.text); // Модель
Car.showInfo(obj);     // Модель ВАЗ-2109
```

Получить значение свойства и вызвать метод можно без создания экземпляра. При использовании инструкции `class` перед именем метода класса указывается ключевое слово `static`. Обратите внимание: внутри статических методов нет доступа к обычным свойствам и методам. Объект мы передаем в качестве параметра статического метода (листинг 3.46).

Листинг 3.46. Статические методы

```
class Car {
   constructor(m) {
      this.model = m;
   }
   static showInfo(obj) {
      console.log(obj.model);
   }
}
var obj = new Car("ВАЗ-2109");
Car.showInfo(obj);     // ВАЗ-2109
```

3.11.7. Перебор свойств объекта

Как видно из листинга 3.42, чтобы обратиться к свойству, следует указать его название после точки. Доступ к методам осуществляется так же, но после имени метода необходимо указать круглые скобки. Кроме точечной нотации, к свойствам и методам можно обратиться как к элементам ассоциативного массива. В этом случае название задается внутри квадратных скобок:

```
var obj1 = new Car("ВАЗ-2109", 2007);
console.log(obj1["year"]);       // 2007
console.log(obj1["getModel"]()); // ВАЗ-2109
```

Обратите внимание на то, что название указывается в виде строки, которую можно изменить внутри программы динамически. Это обстоятельство позволяет обратиться к свойствам, названия которых заранее неизвестны. Выведем названия всех перечислимых свойств и их значения с помощью цикла `for...in` (листинг 3.47).

Листинг 3.47. Перебор свойств объекта

```
function Car(m, y) {
   this.model = m;
   this.year = y;
}
var obj = new Car("ВАЗ-2109", 2007);
for (var prop in obj) {
   // Переменной prop на каждой итерации присваивается
   // название свойства объекта
   console.log(prop + " = " + obj[prop]);
}
```

Перебрать можно только перечислимые свойства, т. е. свойства, для которых метод `propertyIsEnumerable()` из класса `Object` вернул значение `true`. Для свойств встроенных объектов метод возвращает `false`, поэтому перебрать свойства этих объектов с помощью цикла `for...in` нельзя:

```
console.log(obj.propertyIsEnumerable("model")); // true
var x = new Number(10);
console.log(x.propertyIsEnumerable("toFixed")); // false
```

3.11.8. Проверка существования свойств и методов

Оператор `in` также позволяет проверить существование свойства (включая унаследованные) у объекта. Если свойство существует, то возвращается значение `true`:

```
console.log("model" in obj); // true
var y = new Number(5);
console.log("toFixed" in y); // true
```

Проверить наличие не унаследованного свойства позволяет метод `hasOwnProperty()`. В качестве значения указывается название свойства:

```
console.log("toString" in obj);              // true
console.log(obj.hasOwnProperty("toString")); // false
```

Если название метода указать в условии без круглых скобок, то это позволит проверить наличие метода:

```
var z = new Number(5);
if (z.toFixed) console.log("Метод определен");
else console.log("Нет");
```

Обратите внимание на то, что проверять таким образом наличие свойства нельзя, т. к. значение `0` будет интерпретировано как `false`.

С помощью оператора `instanceof` можно проверить принадлежность экземпляра какому-либо объекту:

```
var k = new Number(8);
if ((typeof k == "object") && (k instanceof Number))
console.log("Экземпляр k принадлежит объекту Number");
else console.log("Нет");
```

Удалить свойство позволяет оператор `delete`:

```
console.log("model" in obj);   // true
delete obj.model;
console.log("model" in obj);   // false
```

3.11.9. Прототипы

В предыдущих примерах мы определяли метод `getModel()` внутри конструктора:

```
function Car(m, y) { // Конструктор класса
   this.model = m;
   this.year = y;
   this.getModel = function() {
      return this.model;
   }
}
```

Подобное решение не является эффективным. Предположим, необходимо составить массив, описывающий тысячу автомобилей. Свойства `model` и `year` в этом случае будут содержать разные значения, а вот метод `getModel()` во всех этих объектах один и тот же, но создавать мы его будем тысячу раз.

Использование *прототипов* позволяет определить метод вне конструктора. При создании объекта наследуются все свойства, которые имеются в прототипе. Таким образом, метод `getModel()`, определенный один раз, будет наследоваться всеми экземплярами.

Для добавления метода в прототип используется свойство `prototype` (листинг 3.48).

Листинг 3.48. Прототипы

```
function Car(m, y) {
   this.model = m;
   this.year = y;
}
Car.prototype.getModel = function() {
   return this.model;
}
// Создание экземпляров
var obj1 = new Car("ВАЗ-2109", 2007);
var obj2 = new Car("LADA Granta", 2014);
// Вывод значений
console.log(obj1.year);       // 2007
console.log(obj1.getModel()); // ВАЗ-2109
console.log(obj2.year);       // 2014
console.log(obj2.getModel()); // LADA Granta
```

Как уже говорилось, свойства, определенные в прототипе, наследуются всеми экземплярами. Таким образом, метод `getModel()` доступен для перебора в цикле `for...in`, а также успешно проверяется на наличие с помощью оператора `in`. Тем не менее метод `hasOwnProperty()` позволяет определить, что метод является унаследованным:

```
console.log("getModel" in obj1);              // true
console.log(obj1.hasOwnProperty("getModel")); // false
```

Любой созданный объект автоматически наследует свойства класса `Object`. Например, при попытке вывести значение экземпляра в диалоговом окне, вызывается метод `toString()`, который должен возвращать значение в виде строки. Для примера выведем текущее значение:

```
var obj = new Car("ВАЗ-2109", 2007);
console.log(obj + "");
```

В результате в консоли мы получим следующий результат:

```
[object Object]
```

С помощью прототипов можно переопределить этот метод таким образом, чтобы выводилось нужное нам значение:

```
Car.prototype.toString = function() {
   return "Модель: " + this.model + " Год выпуска: " + this.year;
}
var obj = new Car("ВАЗ-2109", 2007);
console.log(obj + "");
```

В результате в диалоговом окне появится следующий результат:

```
Модель: ВАЗ-2109 Год выпуска: 2007
```

При попытке выполнить арифметическую операцию вызывается метод `valueOf()`, который должен возвращать значение в виде числа (или другого элементарного типа). Для примера переопределим метод таким образом, чтобы он возвращал количество «прожитых» автомобилем лет:

```
Car.prototype.valueOf = function() {
   return (new Date()).getFullYear() - this.year;
}
var obj = new Car("ВАЗ-2109", 2007);
console.log(obj * 1);
```

Практически все встроенные классы JavaScript (например: `String`, `Array`) имеют свойство `prototype`. С его помощью можно расширить возможности встроенных классов — например, добавить новый метод. В качестве примера добавим метод `inArray()` в класс `Array`. Этот метод будет производить поиск значения в массиве и возвращать индекс первого вхождения. Если вхождение не найдено, то метод вернет значение `-1`:

```
Array.prototype.inArray = function(elem) {
   for (var i = 0, len = this.length; i < len; i++) {
      if (this[i] === elem) return i;
   }
   return -1;
}
var arr = [ 1, 2, 3, 4, 5, 1 ];
var pos = arr.inArray(5);
if (pos != -1) console.log("Индекс элемента " + pos);
else console.log("Не найдено");
```

> ***Примечание***
>
> Не рекомендуется расширять возможности встроенных классов, т. к. другие программисты могут прийти в недоумение, увидев новый метод.

3.11.10. Пространства имен

Предположим, программист написал функцию с названием `inArray()`. Через некоторое время ему потребовалось подключить библиотеку стороннего разработчика, в которой все функции объявлены в глобальной области видимости. Если в этой библиотеке объявлена функция с названием `inArray()`, то возникнет конфликт имен. Следует заметить, что никакого сообщения об ошибке в таком случае выведено не будет, — функция, которая объявлена последней, просто переопределит уже существующую функцию. Далее все зависит от частоты использования функции. Все инструкции, которые зависят от этой функции, станут работать некорректно. В итоге будет получен результат, который не планировался, или программа завершится с критической ошибкой.

Чтобы избежать подобной ситуации, следует строго придерживаться *концепции пространств имен*. Согласно этой концепции, модуль может импортировать в гло-

бальную область видимости только один идентификатор. Следует заметить, что это требование касается не только модулей сторонних разработчиков, но и относится к вашим собственным программам.

В языке JavaScript в качестве пространства имен используются объекты. Созданный экземпляр помещается в глобальную область видимости, а остальные идентификаторы доступны через свойства объекта:

```
var myModule = {}; // Объявление пространства имен
myModule.test = function() {
   console.log("Это функция test");
}
myModule.inArray = function() {
   console.log("Это функция inArray");
}
myModule.test();
myModule.inArray();
```

В этом примере функция `inArray()` расположена внутри пространства `myModule`, поэтому конфликт имен сводится к минимуму. Однако может возникнуть ситуация, когда пространства имен называются одинаково. В этом случае решением является создание вложенных объектов. Очень часто название пространства имен совпадает с названием сайта разработчика — в качестве основного объекта используется название зоны, а вложенный объект носит название домена. Например, для сайта **http://example.com/** создание пространства имен будет выглядеть так:

```
var com; // Объявляем, иначе будет ошибка при проверке
if (!com) com = {}; // Объявление пространства имен
else if (typeof com != "object")
   throw new Error("Идентификатор com не является объектом");
if (com.example)
   throw new Error("Пространство имен уже занято");
com.example = { // Объявление вложенного пространства имен
   test: function() {
      console.log("Это функция test");
   },
   inArray: function() {
      console.log("Это функция inArray");
   }
};
com.example.test();
com.example.inArray();
```

Таким образом, если домен принадлежит вам, никакого конфликта имен не будет, но пользоваться таким длинным названием не очень удобно. Учитывая, что присваивание объектов производится по ссылке, а не по значению, эта проблема решается просто — в программе определяется короткий идентификатор и в нем сохраняется ссылка на объект:

```
var $ = com.example;
$.test();
$.inArray();
```

Кроме того, можно использовать анонимную функцию, в параметре которой указывается короткий идентификатор, а при вызове функции передается ссылка на объект, являющийся пространством имен:

```
(function($) {
   $.test();
   $.inArray();
})(com.example);
```

В этом примере идентификатор `$` будет доступен только внутри анонимной функции, а так как функция не имеет названия, в глобальной области видимости никакой идентификатор не сохраняется.

3.12. Обработка ошибок

Существуют три типа ошибок в скриптах: синтаксические, логические и ошибки времени выполнения.

3.12.1. Синтаксические ошибки

Синтаксические — это ошибки в имени оператора или функции, отсутствие закрывающей или открывающей скобок и т. д. То есть ошибки в синтаксисе языка. Как правило, интерпретатор предупредит о наличии ошибки, а программа не будет выполняться совсем.

Например, если вместо:

```
document.write("Строка");
```

написать (здесь нет закрывающей кавычки):

```
document.write("Строка);
```

то Web-браузер в окне консоли отобразит следующее сообщение:

```
SyntaxError: unterminated string literal    index.html:12:15
```

Так Web-браузер предупреждает нас, что существует синтаксическая ошибка. Номер строки с ошибкой (`12`) и название файла (`index.html`) отображаются в правой части строки. Достаточно отсчитать 12 строк в исходном коде и добавить закрывающую кавычку. А затем обновить страницу.

Приведем часто встречающиеся синтаксические ошибки:

- опечатка в имени оператора или функции;
- буква набрана в русской раскладке клавиатуры вместо латинской;
- неправильный регистр букв;

- отсутствие открывающей или закрывающей скобки (или, наоборот, лишние скобки);
- в цикле `for` указаны параметры через запятую, а не через точку с запятой.

3.12.2. Логические ошибки

Логические ошибки — это ошибки в логике работы программы, которые можно выявить только по полученному результату. Такие ошибки весьма трудно выявить и исправить. Как правило, интерпретатор не предупреждает о наличии логической ошибки, и программа будет выполняться, т. к. не содержит синтаксических ошибок.

Предположим, необходимо вывести первые три элемента массива. Программист, забыв, что нумерация массивов начинается с нуля, пишет следующий код:

```
var arr = [1, 2, 3, 4];
for (var i = 1; i < 4; i++) console.log(arr[i]);
```

В итоге возникает логическая ошибка, поскольку будут получены не первые элементы массива, а три элемента, начиная со второго. Но так как в этом примере нет синтаксических ошибок, интерпретатор сочтет код правильным.

Если в логическом выражении вместо оператора `==` (равно) указан оператор присваивания `=`, то это также приведет к логической ошибке:

```
var x = 5;
if (x = 6) console.log("Переменная x равна 6");
else console.log("Переменная x не равна 6");
```

Этот код выведет совсем не то, что хотел программист:

```
Переменная x равна 6
```

3.12.3. Ошибки времени выполнения

Ошибки времени выполнения — это ошибки, которые возникают во время работы скрипта. Причиной являются события, не предусмотренные программистом.

В некоторых языках (например, в PHP) ошибки времени выполнения возникают из-за деления на ноль или обращения к несуществующему элементу массива. Но в языке JavaScript в этих случаях программа прервана не будет — при попытке деления на ноль возвращается значение `Infinity` или `NaN`:

```
console.log(5 / 0); // Infinity
console.log(0 / 0); // NaN
```

При обращении к несуществующему элементу массива возвращается значение `undefined`:

```
var arr = [ 1, 2 ];
console.log(arr[20]); // undefined
```

Очень часто ошибки времени выполнения возникают при использовании условий:

```
var x = 6;
if (x > 5) console.log("x > 5");
else conole.log(x); // Строка с ошибкой ( conole вместо console)
```

В этом примере никакой ошибки не будет, пока соблюдается условие `"x > 5"`. Как только условие перестанет выполняться, сразу возникнет ошибка, и выполнение программы будет прервано.

3.12.4. Обработка ошибок

Перехватить и обработать ошибки позволяет конструкция `try...catch`. Конструкция имеет следующий формат:

```
try {
   <Инструкции, в которых перехватываем ошибки>
}
[catch ([<Ссылка на объект Error>]) {
   <Обработка ошибки>
}]
[finally {
   <Инструкции, которые будут выполнены в любом случае>
}]
```

Инструкции, в которых могут возникнуть ошибки, размещаются в блоке `try`. Если внутри этого блока возникнет исключение, то управление будет передано в блок `catch`. В качестве параметра в блоке `catch` можно указать переменную, через которую будет доступен объект `Error`, содержащий описание ошибки. Если в блоке `try` ошибки не возникло, то блок `catch` не выполняется. Если указан блок `finally`, то инструкции внутри этого блока будут выполнены независимо от того, возникла ошибка или нет. Блоки `catch` и `finally` не являются обязательными, но хотя бы один из них должен присутствовать:

```
var x = 3;
try {
   if (x > 5) console.log("x > 5");
   else conole.log(x); // Строка с ошибкой
} catch (e) {
   console.log("Возникла ошибка: " + e.name + ": " + e.message);
} finally {
   console.log("Сообщение будет выведено в любом случае");
}
```

Результат в консоли при ошибке:

```
Возникла ошибка: ReferenceError: conole is not defined
Сообщение будет выведено в любом случае
```

3.12.5. Оператор *throw*. Генерация исключений

В некоторых случаях требуется не обрабатывать ошибку, а, наоборот, указать программе, что возникла неисправимая ошибка, и прервать выполнение всей программы или передать управление вышестоящему обработчику. Для этого предназначен оператор `throw`:

```
var x = -3;
try {
   if (x < 0)
      throw new Error("Переменная не может быть меньше нуля");
} catch (e) {
   console.log("Возникла ошибка: " + e.name + ": " + e.message);
}
```

3.12.6. Отладка программы в Web-браузере Firefox

Как вы уже знаете, Web-браузер Firefox содержит панель **Инструменты разработчика**. В состав этого средства входит **Веб-консоль**, в которую выводятся различные сообщения — например, при наличии ошибок в программе. Чтобы открыть эту консоль, переходим в главное меню и выбираем пункт **Разработка | Веб-консоль** или нажимаем комбинацию клавиш <Ctrl>+<Shift>+<K>. В результате отобразится содержимое вкладки **Консоль**.

Консолью мы уже не раз пользовались при выводе результатов работы учебных примеров. Сейчас нас интересует вкладка **Отладчик**. Так что переходим в главное меню и выбираем пункт **Разработка | Отладчик** или нажимаем комбинацию клавиш <Ctrl>+<Shift>+<S>.

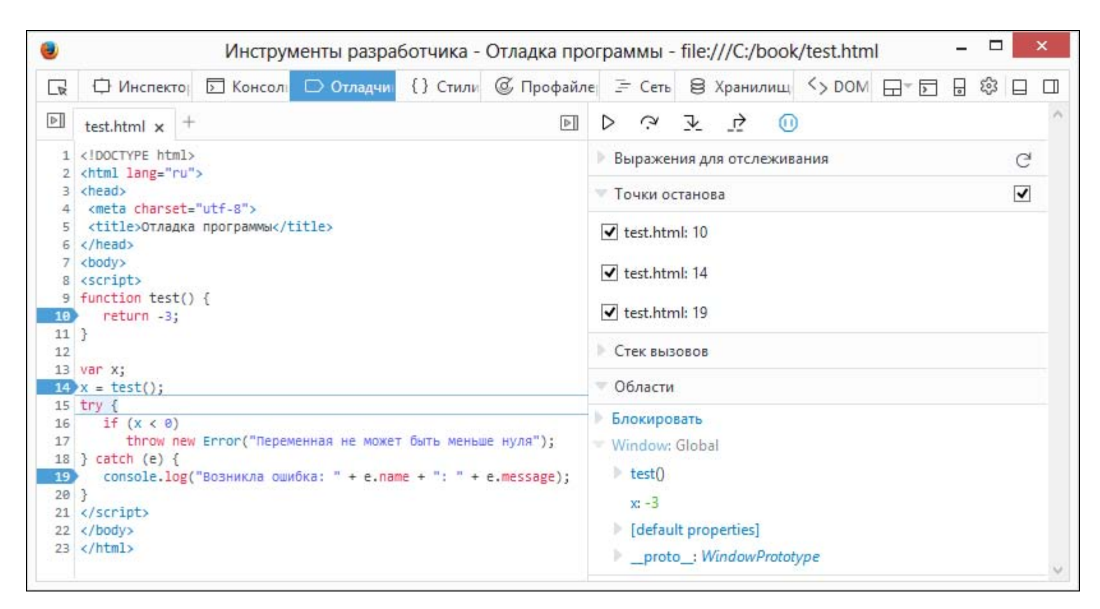


Рис. 3.1. Пошаговое выполнение программы на вкладке **Отладчик**

Вкладка **Отладчик** является полноценным *отладчиком* скриптов на JavaScript. На этой вкладке существует возможность выполнять программу по шагам, отслеживая значения различных переменных, — это очень удобное средство для поиска ошибок в программе, особенно логических. Вначале нужно установить *точки останова* — для чего следует щелкнуть мышью напротив нужной строки на нумерации строк. Теперь после обновления Web-страницы программа прервется на отмеченной строке (рис. 3.1). В этот момент можно посмотреть текущие значения переменных, а также продолжить выполнение скрипта по шагам. Так можно полностью контролировать весь процесс выполнения программы. Пользуйтесь отладкой при возникновении любой ошибки, и вы очень быстро ее найдете и исправите.

3.13. События

При взаимодействии пользователя с Web-страницей происходят события. *События* — это своего рода извещения системы о том, что пользователь выполнил какое-либо действие, или внутри самой системы возникло некоторое условие. События возникают при щелчке на элементе, перемещении мыши, нажатии клавиши на клавиатуре, изменении размеров окна, окончании загрузки Web-страницы и т. д.

3.13.1. Назначение обработчиков событий

Зная, какие события может генерировать тот или иной элемент Web-страницы, можно написать функцию для обработки этого события. Например, при отправке данных формы возникает событие `onsubmit`. При наступлении этого события можно проверить данные, введенные пользователем, и, если они не соответствуют ожидаемым, прервать отправку данных.

Назначить обработчик события можно несколькими способами. Первый способ заключается в добавлении к тегам параметров с названиями событий. В качестве значений параметров указывается выражение или вызов функции.

Вот пример назначения обработчика нажатия кнопки:

```
<input type="button" value="Кнопка 1" onclick="handler1()">
```

При нажатии кнопки будет вызвана функция `handler1()`.

Второй способ подразумевает использование свойств элементов. Причем тут возможны два варианта: с помощью анонимной функции или ссылки на функцию:

```
document.getElementById("btn2").onclick = function() {
   console.log("Нажата кнопка 2");
};
document.getElementById("btn3").onclick = handler2;
```

Чтобы получить доступ к свойствам элемента, нужно вначале получить ссылку на сам элемент с помощью метода `getElementById()` объекта `document`. В качестве значения метод принимает строку с идентификатором элемента. Далее указывается название события, которое совпадает с названием параметра тега. После оператора `=` задается анонимная функция или ссылка на функцию. Обратите внимание: при

указании ссылки название функции указывается без круглых скобок. Если круглые скобки указать, то функция будет вызвана, и результат ее работы станет значением свойства.

Описанные два способа позволяют назначить только один обработчик. Попытка присвоить другое значение свойству приведет к удалению имеющегося обработчика. Третий способ лишен этого недостатка. Назначить обработчик события позволяет метод `addEventListener()`. Формат метода:

```
addEventListener(<Событие>, <Ссылка на функцию>[, <Фаза>]);
```

В параметре `<Событие>` указывается название события в виде строки без префикса `on` — например, `click` вместо `onclick`. Ссылка на функцию-обработчик указывается во втором параметре. В эту функцию в качестве параметра передается ссылка на объект `event`, а внутри функции через ключевое слово `this` доступна ссылка на текущий элемент. Описание параметра `<Фаза>` см. в *разд. 3.13.7*.

Пример назначения обработчиков различными способами приведен в листинге 3.49.

Листинг 3.49. Назначение обработчиков событий

```
<!DOCTYPE html>
<html lang="ru">
<head>
 <meta charset="utf-8">
 <title>Назначение обработчиков событий</title>
</head>
<body>
<input type="button" value="Кнопка 1" onclick="handler1()">
<input type="button" value="Кнопка 2" id="btn2">
<input type="button" value="Кнопка 3" id="btn3">
<input type="button" value="Кнопка 4" id="btn4">
<script>
function handler1() {
   console.log("Нажата кнопка 1");
}
function handler2() {
   console.log("Нажата кнопка 3");
}
function handler3(e) {
   console.log("Нажата кнопка 4. handler3()");
}
function handler4(e) {
   console.log("Нажата кнопка 4. handler4()");
}
document.getElementById("btn2").onclick = function() {
   console.log("Нажата кнопка 2");
};
```

```
// Название функции указывается без круглых скобок
document.getElementById("btn3").onclick = handler2;
// Можно назначить сразу несколько обработчиков
var btn4 = document.getElementById("btn4");
btn4.addEventListener("click", handler3);
btn4.addEventListener("click", handler4);
</script>
</body>
</html>
```

> **Примечание**
>
> В старых версиях браузера Internet Explorer метод `addEventListener()` не работает. Вместо него нужно использовать метод `attachEvent()`. А удаление обработчика осуществляется с помощью метода `detachEvent()`.

3.13.2. Удаление обработчиков

Если обработчик назначался через параметр тега или свойство, то для удаления обработчика нужно присвоить свойству значение `null`, пустую строку или пустую анонимную функцию (последний способ работает во всех Web-браузерах):

```
document.getElementById("btn2").onclick = function() {};
```

Если обработчик назначался через метод `addEventListener()`, то удалить его можно с помощью метода `removeEventListener()`. Формат метода:

```
removeEventListener(<Событие>, <Ссылка на функцию>[, <Фаза>]);
```

Обратите внимание: нужно обязательно иметь ссылку на обработчик, назначенный с помощью метода `addEventListener()`:

```
document.getElementById("btn4").removeEventListener("click", handler3);
```

Если при назначении обработчика использовалась анонимная функция, то удалить обработчик будет нельзя.

3.13.3. Указатель *this*

При назначении обработчика с помощью свойства или метода `addEventListener()`, внутри обработчика будет доступна ссылка на текущий элемент через указатель `this`. С помощью этого указателя можно получить доступ к свойствам элемента. Если обработчик назначается через параметр тега, то указатель нужно передать в качестве параметра. Получим текст на кнопке по ее нажатию (листинг 3.50).

Листинг 3.50. Указатель `this`

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
```

```
   <title>Указатель this</title>
</head>
<body>
<input type="button" value="Кнопка 1" onclick="handler1(this)">
<input type="button" value="Кнопка 2" id="btn2">
<input type="button" value="Кнопка 3" id="btn3">
<script>
function handler1(elem) {
   console.log("Нажата кнопка " + elem.value);
}
function handler2(event) {
   console.log("Нажата кнопка " + this.value);
}
document.getElementById("btn2").onclick = function(event) {
   console.log("Нажата кнопка " + this.value);
};
var btn3 = document.getElementById("btn3");
btn3.addEventListener("click", handler2);
</script>
</body>
</html>
```

3.13.4. Объект *event*

Объект `event` позволяет получить детальную информацию о произошедшем событии и выполнить необходимые действия. Объект `event` доступен только в обработчиках событий. При наступлении следующего события все предыдущие значения свойств сбрасываются.

При назначении обработчика с помощью метода `addEventListener()` или свойства объект `event` будет доступен через первый параметр. Если обработчик назначается через параметр тега, то объект нужно передать в обработчик в качестве параметра.

Объект `event` имеет следующие основные свойства:

- `type` — строка, содержащая тип события. Возвращается в нижнем регистре и без префикса `on`. Например, при событии `onclick` свойство `type` равно `click`;
- `target` — ссылка на элемент, который является источником события (в старых версиях браузера Internet Explorer используется свойство `srcElement`);
- `currentTarget` — возвращает ссылку на элемент, в котором обрабатывается событие. Ссылается на тот же элемент, что и ключевое слово `this` внутри обработчика события. Значение свойства `currentTarget` может не совпадать со значением свойства `target`;
- `timeStamp` — время возникновения события в миллисекундах.

Благодаря объекту `event`, мы можем назначить один обработчик сразу для нескольких элементов и даже для нескольких типов событий. В листинге 3.51 приведен пример обработки нажатия кнопок внутри одного обработчика.

Листинг 3.51. Объект event

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Объект event</title>
</head>
<body>
<input type="button" value="Кнопка 1" onclick="handler(event)">
<input type="button" value="Кнопка 2" id="btn2">
<input type="button" value="Кнопка 3" id="btn3">
<script>
function handler(event) {
   console.log(event.type);
   console.log(event.target.value);
   console.log(event.currentTarget.value);
   console.log(event.timeStamp);
}
document.getElementById("btn2").onclick = handler;
var btn3 = document.getElementById("btn3");
btn3.addEventListener("click", handler);
</script>
</body>
</html>
```

3.13.5. Действия по умолчанию и их отмена

Для многих событий назначены действия по умолчанию, т. е. действия, которые Web-браузер выполняет в ответ на возникшие в документе события. Например, при щелчке на гиперссылке действием по умолчанию будет переход по указанному URL-адресу, нажатие кнопки **Отправить** приводит к отправке данных формы и т. д.

Иногда действия по умолчанию необходимо прервать. Например, при отправке данных формы можно проверить их на соответствие ожидаемым и, если они не соответствуют, прервать отправку. Для этого используются следующие свойства и методы объекта `event`:

- `cancelable` — содержит `true`, если действие по умолчанию может быть отменено, и `false` — в противном случае;
- `preventDefault()` — отменяет действие по умолчанию, если его можно отменить. В браузере Internet Explorer для отмены служит свойство `returnValue`, которому нужно присвоить значение `false`;
- `defaultPrevented` — содержит `true`, если действие по умолчанию для текущего события было отменено в этом или предыдущем обработчике, и `false` — в противном случае.

Для отмены действия по умолчанию можно также внутри обработчика вернуть значение `false`.

В листинге 3.52 приведен пример проверки правильности ввода E-mail и прерывания перехода по гиперссылке.

Листинг 3.52. Прерывание действий по умолчанию

```
<!DOCTYPE html>
<html lang="ru">
<head>
 <meta charset="utf-8">
 <title>Прерывание действий по умолчанию</title>
<script>
function testEmail() {
   var p = /^[a-z0-9_.-]+@([a-z0-9-]+\.)+[a-z]{2,6}$/i;
   // Получаем значение поля email
   var email = document.getElementById("email").value;
   if (p.test(email)) {
      if (window.confirm("Отправить данные формы?")) {
         return true;    // Отправляем
      }
      else return false; // Прерываем
   }
   else {
      alert("E-mail введен неправильно");
      return false;      // Прерываем
   }
}
function handler(e) {
   e = e || window.event;
   if (e.preventDefault) e.preventDefault();
   else e.returnValue = false;
   alert("Перехода по ссылке не будет!");
}
</script>
</head>
<body>
<form action="file.php" method="GET" onsubmit="return testEmail()">
   <div>
   E-mail:<br>
   <input type="text" name="email" id="email"><br>
   <input type="submit" value="Отправить">
   </div>
</form>
<p>
<a href="file.html"
```

```
  onclick="alert('Перехода по ссылке не будет!'); return false">
  Нажмите для перехода по ссылке</a><br><br>
<a href="file.html" onclick="handler(event)">
  Нажмите для перехода по ссылке</a>
</p>
</body>
</html>
```

В этом примере рассмотрены два способа прерывания действия по умолчанию: в первой ссылке прерывание действия по умолчанию осуществляется возвратом значения `false`, а во второй — с помощью свойств и методов объекта `event`.

3.13.6. Всплывание событий

Что же такое *всплывание событий*? Давайте рассмотрим следующий пример (листинг 3.53).

Листинг 3.53. Всплывание событий

```
<!DOCTYPE html>
<html lang="ru">
<head>
 <meta charset="utf-8">
 <title>Всплывание событий</title>
<script>
function showMsg(msg) {
   var div1 = document.getElementById("div1");
   div1.innerHTML += msg + "<br>";
}
</script>
</head>
<body onclick="showMsg('Событие onclick - Документ')">
<p onclick="showMsg('Событие onclick - Абзац')">
Щелкните мышью
<span style="color: red" onclick="showMsg('Событие onclick - SPAN')">
здесь</span>
</p>
<div id="div1"></div>
</body>
</html>
```

В этом примере мы написали обработчики события `onclick` для трех элементов страницы: тела документа, абзаца и тега `<span>`. Попробуем щелкнуть левой кнопкой мыши на слове `здесь` — вместо одного события `onclick` мы получим целую последовательность событий:

```
Событие onclick - SPAN
Событие onclick - Абзац
Событие onclick - Документ
```

Иными словами, событие `onclick` последовательно передается элементу-родителю. Для тега `<span>` элементом-родителем является абзац. А для абзаца элементом-родителем является само тело документа. Такое прохождение событий и называется их *всплыванием*.

Управлять всплыванием события позволяют следующие свойства и методы объекта `event`:

- `bubbles` — содержит `true`, если текущее событие может всплывать, и `false` — в противном случае;
- `stopPropagation()` — прерывает всплывание события. В браузере Internet Explorer для прерывания используется свойство `cancelBubble` объекта `event`, которому нужно присвоить значение `true`.

Продемонстрируем прерывание всплывания события на примере (листинг 3.54).

Листинг 3.54. Прерывание всплывания события

```
<!DOCTYPE html>
<html lang="ru">
<head>
 <meta charset="utf-8">
 <title>Прерывание всплывания события</title>
<script>
function showMsg(e, msg) {
   var div1 = document.getElementById("div1");
   div1.innerHTML += msg + "<br>";
   e = e || window.event;
   if (e.stopPropagation) e.stopPropagation();
   else e.cancelBubble = true;
}
</script>
</head>
<body onclick="showMsg(event, 'Событие onclick - Документ')">
<p onclick="showMsg(event, 'Событие onclick - Абзац')">
Щелкните мышью
<span style="color: red"
      onclick="showMsg(event, 'Событие onclick - SPAN')">здесь</span>
</p>
<div id="div1"></div>
</body>
</html>
```

Попробуем теперь щелкнуть левой кнопкой мыши на слове `здесь` — вместо трех событий мы получим только одно:

```
Событие onclick - SPAN
```

3.13.7. Фазы событий

Помимо *фазы всплывания события* существует также *фаза перехвата события*. Эта фаза не работает в старых версиях браузера Internet Explorer, поэтому используется редко. Узнать текущую фазу позволяет свойство `eventPhase` объекта `event`. Оно может содержать следующие значения:

- `1` — событие было перехвачено у вложенного элемента;
- `2` — событие в настоящий момент обрабатывается в том же элементе, в котором и возникло;
- `3` — событие всплыло из вложенного элемента.

При изучении метода `addEventListener()` мы оставили без внимания параметр `<Фаза>`. Чтобы понять смысл этого параметра, рассмотрим пример (листинг 3.55).

Листинг 3.55. Фазы событий

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Фазы событий</title>
</head>
<body>
<div><span id="span1">span1
     <span id="span2">Щелкните здесь (span2)</span></span>
</div>
<script>
function handler(e) {
   console.log("Элемент " + this.id +
               ". Событие возникло в " + e.target.id);
   console.log(e.eventPhase);
}
if (document.addEventListener) {
   var span1 = document.getElementById("span1");
   var span2 = document.getElementById("span2");
   span1.addEventListener("click", handler, true);
   span2.addEventListener("click", handler, false);
}
</script>
</body>
</html>
```

При щелчке на ссылке `Щелкните здесь` возникнет последовательность событий:

```
Элемент span1. Событие возникло в span2
Элемент span2. Событие возникло в span2
```

Таким образом, событие, возникшее во вложенном элементе, вначале обрабатывается элементом-родителем, а затем самим элементом. Если заменить `true` на `false`, то последовательность будет другой:

```
Элемент span2. Событие возникло в span2
Элемент span1. Событие возникло в span2
```

Это нормальная последовательность всплывания событий, которую мы рассматривали в *разд. 3.13.6*. Именно значение `false` используется в большинстве случаев.

3.13.8. События документа

Приведем основные события документа:

- `onload` — после полной загрузки Web-страницы. Внутри обработчика этого события можно обращаться ко всем элементам из скрипта и назначать обработчики событий для элементов страницы:

  ```
  window.onload = function() {
     // Здесь назначаем обработчики событий для элементов
  };
  ```

- `onscroll` — при прокручивании содержимого элемента страницы, документа, окна или фрейма;
- `onresize` — при изменении размеров окна;
- `onbeforeunload` — перед выгрузкой документа. Если внутри обработчика вернуть строку, то Web-браузер выведет окно, в котором пользователю будет предложено остаться;
- `onunload` — непосредственно перед выгрузкой документа. Генерируется после события `onbeforeunload`;
- `onbeforeprint` — перед распечаткой документа или вывода его на предварительный просмотр;
- `onafterprint` — после распечатки документа или вывода его на предварительный просмотр.

Пример обработки событий документа приведен в листинге 3.56.

Листинг 3.56. События документа

```
<!DOCTYPE html>
<html lang="ru">
<head>
 <meta charset="utf-8">
 <title>События документа</title>
<script>
function showMsg(event, msg) {
   var div1 = document.getElementById("div1");
   div1.innerHTML += msg + "<br>";
```

```
   console.log(event);
}
window.onbeforeunload = function() {
   return "Вы действительно хотите уйти?";
};
</script>
</head>
<body onload="showMsg(event, 'Событие onload')"
      onscroll="showMsg(event, 'Событие onscroll')"
      onresize="showMsg(event, 'Событие onresize')"
      onbeforeprint="showMsg(event, 'Событие onbeforeprint')"
      onafterprint="showMsg(event, 'Событие onafterprint')">
<div id="div1"></div>
<p><a href="test.html">Ссылка</a></p>
<div style="height: 600px"></div>
</body>
</html>
```

Попробуйте перейти по ссылке или закрыть окно Web-браузера — откроется окно с запросом. Событие `onbeforeunload`, возникающее перед выгрузкой документа, позволяет вывести стандартное диалоговое окно с двумя кнопками. Окно может содержать текст, возвращенный из обработчика, но большинство Web-браузеров выводит стандартное сообщение.

3.13.9. События мыши

Приведем основные события мыши:

- `onmousedown` — при нажатии кнопки мыши на элементе Web-страницы или самой странице;
- `onmouseup` — при отпускании ранее нажатой кнопки мыши;
- `onclick` — при щелчке мыши на элементе или на Web-странице;
- `ondblclick` — при двойном щелчке мыши;
- `onmousemove` — при любом перемещении мыши;
- `onmouseover` — при наведении курсора мыши на элемент;
- `onmouseout` — при выведении курсора мыши с элемента;
- `onselect` — при выделении элемента;
- `oncontextmenu` — при нажатии правой кнопки мыши для вывода контекстного меню;
- `onwheel` — при вращении колесика мыши. В старых Web-браузерах используется событие `onmousewheel`.

События возникают последовательно — например, последовательность событий при нажатии кнопки мыши на элементе страницы будет такой:

```
onmousedown
onmouseup
onclick
```

А при двойном нажатии — такой:

```
onmousedown
onmouseup
onclick
ondblclick
```

Это значит, что событие `ondblclick` возникает после события `onclick`.

Продемонстрируем обработку событий мыши на примере (листинг 3.57).

Листинг 3.57. События мыши

```
<!DOCTYPE html>
<html lang="ru">
<head>
 <meta charset="utf-8">
 <title>События мыши</title>
<script>
function showMsg(event, msg) {
   var div1 = document.getElementById("div1");
   div1.innerHTML += msg + "<br>";
   console.log(event);
}
</script>
</head>
<body onmousedown="showMsg(event, 'Событие onmousedown')"
      onmouseup="showMsg(event, 'Событие onmouseup')"
      onclick="showMsg(event, 'Событие onclick')"
      ondblclick="showMsg(event, 'Событие ondblclick')"
      oncontextmenu="showMsg(event, 'Событие oncontextmenu')">
<p onmouseover="showMsg(event, 'Событие onmouseover')"
   onmouseout="showMsg(event, 'Событие onmouseout')">
Щелкните мышью в любом месте страницы
</p>
<div id="div1"></div>
<div style="height: 600px"></div>
</body>
</html>
```

Получить информацию о событии позволяют следующие свойства:

- `clientX` и `clientY` — координаты события (по осям `X` и `Y`) в клиентских координатах;
- `pageX` и `pageY` — координаты события (по осям `X` и `Y`) относительно левого верхнего угла страницы;

- offsetX и offsetY — координаты события (по осям X и Y) относительно контейнера;
- button — число, указывающее нажатую кнопку мыши. Может принимать следующие значения (в старых версиях браузера Internet Explorer используются другие значения):
  - 0 — нажата левая кнопка мыши;
  - 1 — нажата средняя кнопка;
  - 2 — нажата правая кнопка мыши;
- relatedTarget — для события onmouseover содержит ссылку на элемент, с которого переместился курсор мыши. Для события onmouseout содержит ссылку на элемент, на который пользователь перемещает курсор мыши (в браузере Internet Explorer используются свойства fromElement и toElement);
- detail — для событий onclick, onmousedown и onmouseup возвращает количество выполненных щелчков мышью;
- deltaX, deltaY и deltaZ — позволяют определить направление вращения колесика мыши при событии onwheel.

3.13.10. События клавиатуры

Приведем события клавиатуры:

- onkeydown — при нажатии клавиши на клавиатуре;
- onkeypress — аналогично событию onkeydown, но генерируется только для символьных клавиш;
- onkeyup — при отпускании ранее нажатой клавиши клавиатуры.

При нажатии клавиши на клавиатуре последовательность будет такой:

```
onkeydown
onkeypress
onkeyup
```

Продемонстрируем обработку событий клавиатуры на примере (листинг 3.58).

Листинг 3.58. События клавиатуры

```
<!DOCTYPE html>
<html lang="ru">
<head>
 <meta charset="utf-8">
 <title>События клавиатуры</title>
<script>
function showMsg(event, msg) {
   var div1 = document.getElementById("div1");
   div1.innerHTML += msg + "<br>";
```

```
    console.log(event);
}
</script>
</head>
<body onkeydown="showMsg(event, 'Событие onkeydown')"
      onkeypress="showMsg(event, 'Событие onkeypress')"
      onkeyup="showMsg(event, 'Событие onkeyup')">
<p>Нажмите клавишу на клавиатуре</p>
<div id="div1"></div>
</body>
</html>
```

Получить информацию о событии позволяют следующие свойства:

- key — строка с буквой или описанием клавиши (например, "ArrowDown");
- keyCode — код нажатой клавиши. В Web-браузере Firefox при нажатии обычной клавиши в обработчике события onkeypress свойство keyCode имеет значение 0, а код символа доступен через свойство charCode. Если нажата только функциональная клавиша, то ситуация другая — свойство charCode имеет значение 0, а код символа доступен через свойство keyCode;
- location — позволяет определить, на какой клавиатуре была нажата клавиша:
  - 0 — на обычной клавиатуре;
  - 1 — функциональная клавиша (например, <Ctrl>) нажата слева;
  - 2 — функциональная клавиша (например, <Ctrl>) нажата справа;
  - 3 — на цифровой клавиатуре;
- repeat — true, если клавиша удерживается нажатой;
- shiftKey — true, если была нажата клавиша <Shift>;
- ctrlKey — true, если была нажата клавиша <Ctrl>;
- altKey — true, если была нажата клавиша <Alt>;
- metaKey — true, если была нажата клавиша <Meta>.

3.13.11. События формы

Приведем основные события формы:

- onsubmit — при отправке данных формы;
- onreset — при очистке формы;
- onfocus — при получении фокуса элементом формы;
- onchange — при изменении данных в текстовом поле и перемещении фокуса на другой элемент формы либо при отправке данных формы (наступает перед событием onblur);
- onblur — при потере фокуса элементом формы;

- `oninput` — периодически возникает в процессе ввода данных в поле ввода или в область редактирования;
- `oninvalid` — возникает, например, если не заполнено обязательное поле.

Продемонстрируем обработку событий формы на примере (листинг 3.59).

Листинг 3.59. События формы

```
<!DOCTYPE html>
<html lang="ru">
<head>
 <meta charset="utf-8">
 <title>События формы</title>
<script>
function showMsg(event, msg) {
   var div1 = document.getElementById("div1");
   div1.innerHTML += msg + "<br>";
   console.log(event);
}
</script>
</head>
<body>
 <form action="file.php" method="GET"
       onsubmit="showMsg(event, 'Событие onsubmit'); return false"
       onreset="showMsg(event, 'Событие onreset')">
  <div>
   Логин:<br>
   <input type="text" name="login"
          onfocus="showMsg(event, 'Событие onfocus')"
          onblur="showMsg(event, 'Событие onblur')"
          onchange="showMsg(event, 'Событие onchange')"
          oninput="showMsg(event, 'Событие oninput')"><br>
   E-mail:<br>
   <input type="text" name="email" required
          oninvalid="showMsg(event, 'Событие oninvalid')"><br>
   Описание:<br>
   <textarea name="descr" rows="10" cols="15"></textarea><br>
   <input type="reset" value="Очистить">
   <input type="submit" value="Отправить">
  </div>
 </form>
<div id="div1"></div>
</body>
</html>
```

Нужно еще раз напомнить, что примеры эти написаны только для браузера Firefox. Например, в Web-браузере Internet Explorer половина приведенных свойств не

работает. Поэтому для написания скриптов, которые будут правильно работать во всех Web-браузерах, приходится писать код под каждый Web-браузер отдельно. Также нужно учитывать и версию Web-браузера.

3.14. Объектная модель документа (*DOM*)

Объектная модель документа — это совокупность объектов, обеспечивающих доступ к содержимому Web-страницы и ряду функций Web-браузера. Доступ к объектам позволяет управлять содержимым Web-страницы уже после ее загрузки.

3.14.1. Структура объектной модели

Объектная модель представлена в виде иерархии объектов. То есть имеется объект верхнего уровня и подчиненные ему объекты. В свою очередь, подчиненные объекты имеют свои подчиненные объекты. Кроме того, все объекты имеют свойства, а некоторые еще и методы.

Часто объект верхнего уровня (и даже подчиненный объект) можно не указывать. Давайте в качестве примера рассмотрим инструкцию для вызова диалогового окна с сообщением. Это окно мы не раз использовали для вывода результата работы скрипта:

```
window.alert("Сообщение");
```

Здесь `window` — это объект самого верхнего уровня, представляющий непосредственно Web-браузер, а `alert()` — это метод объекта `window`. В этом случае указывать объект не обязательно, т. к. объект `window` подразумевается по умолчанию:

```
alert("Сообщение");
```

Кстати, мы не раз опускали упоминание объекта верхнего уровня. Например, при печати сообщения в окне Web-браузера:

```
document.write("Сообщение");
```

Поскольку объект `document` является подчиненным объекту `window`, то нужно было бы написать так:

```
window.document.write("Сообщение");
```

Помимо уже упомянутого объекта самого высокого уровня — `window` — в объектной модели имеются следующие основные объекты:

- `frames` — коллекция фреймов (см. *разд. 3.14.3*);
- `navigator` — содержит информацию о Web-браузере (см. *разд. 3.14.4*);
- `screen` — служит для доступа к характеристикам экрана компьютера пользователя (см. *разд. 3.14.5*);
- `location` — содержит URL-адрес текущей Web-страницы (см. *разд. 3.14.6*);
- `history` — предоставляет доступ к списку истории Web-браузера (см. *разд. 3.14.7*);

- `document` — служит для доступа к структуре и содержанию документа:
  - `forms` — коллекция всех форм;
  - `images` — коллекция всех изображений;
  - `links` — коллекция ссылок;
  - `scripts` — коллекция скриптов.

3.14.2. Объект *window*

Объект `window` — это объект самого верхнего уровня, представляющий окно Web-браузера. Объект `window` подразумевается по умолчанию, поэтому указывать ссылку на этот объект не обязательно.

Свойства объекта `window`:

- `status` — сообщение, отображаемое в строке состояния:

  ```
  window.status = "Текст в строке состояния";
  ```

- `screenX` — горизонтальная координата левого верхнего угла окна;
- `screenY` — вертикальная координата левого верхнего угла окна;
- `outerWidth` — полная ширина окна в пикселах;
- `outerHeight` — полная высота окна в пикселах;
- `innerWidth` — ширина содержимого окна в пикселах;
- `innerHeight` — высота содержимого окна в пикселах. Размер не включает высоту строки меню и панели инструментов;
- `scrollX` и `pageXOffset` — число пикселов, на которое документ прокручен по горизонтали;
- `scrollY` и `pageYOffset` — число пикселов, на которое документ прокручен по вертикали.

Методы объекта `window`:

- `alert()` — отображает окно сообщения (см. *разд. 3.1.6*);
- `confirm()` — выводит окно подтверждения (см. *разд. 3.1.7*);
- `prompt()` — показывает окно с полем ввода (см. *разд. 3.1.8*);
- `scrollTo(<X>, <Y>)` — прокручивает содержимое окна в точку с координатами <X> и <Y>;
- `scrollBy(<tX>, <tY>)` — прокручивает окно на заданные расстояния;
- `open()` и `showModalDialog()` — открывают новое окно. Первый метод открывает обычное окно, а второй — модальное (блокирующее другие окна до момента закрытия). Так как всплывающие окна стали использовать для отображения рекламы, Web-браузеры добавили возможность блокировки всплывающих окон в настройках (метод `showModalDialog()` многие Web-браузеры вообще удалили),

и эта блокировка устанавливается по умолчанию. Поэтому рассчитывать, что пользователь увидит окно, не стоит. На сегодняшний день все Web-страницы отображаются на отдельных вкладках, а не в отдельных окнах, поэтому открывать отдельное окно, а тем более модальное, — это не уважать пользователя. Для имитации создания диалогового окна лучше использовать возможности JavaScript-библиотек — например, jQuery UI Dialog.

Кроме того, имеются четыре метода для работы с таймерами, которые мы рассмотрели в *разд. 3.9.6*.

3.14.3. Работа с фреймами

Для работы с фреймами предназначены следующие свойства объекта `window`:

- `frames` — коллекция фреймов. Обратиться к элементам коллекции можно по индексу, указанному внутри квадратных скобок, а также по названию (значению параметра `name` тега `<iframe>`). Общее количество фреймов доступно через свойство `length`:

```
console.log(window.frames.length);
console.log(window.frames[0].document.title);
console.log(window.frames.frame1.document.title);
```

- `length` — количество фреймов, отображаемых в текущем окне;
- `name` — имя окна или фрейма;
- `parent` — ссылка на родительское окно:

```
console.log("parent: " + window.parent.document.title);
```

- `self` и `window` — ссылка на текущее окно:

```
console.log("self: " + window.self.document.title);
```

- `top` — ссылка на самое верхнее родительское окно:

```
console.log("top: " + window.top.document.title);
```

Обратите внимание на то, что в целях безопасности документ, загруженный с другого домена, не имеет доступа к родительскому окну.

3.14.4. Объект *navigator*. Получение информации о Web-браузере

Объект `navigator` предоставляет информацию о самом Web-браузере.

Свойства объекта `navigator`:

- `userAgent` — строка, содержащая полную информацию о Web-браузере (пользователь в настройках может изменить информацию):

```
console.log(window.navigator.userAgent);
// Mozilla/5.0 (Windows NT 6.2; Win64; x64; rv:56.0)
// Gecko/20100101 Firefox/56.0
```

- `appCodeName` — название Web-браузера:

```
console.log(navigator.appCodeName);   // Mozilla
```

- `appName` — внутреннее имя Web-браузера:

```
console.log(navigator.appName);       // Netscape
```

- `appVersion` — внутренняя версия Web-браузера:

```
console.log(navigator.appVersion);    // 5.0 (Windows)
```

- `platform` — название клиентской платформы:

```
console.log(navigator.platform);      // Win64
```

- `language` — предпочитаемый пользователем язык:

```
console.log(navigator.language);      // ru-RU
```

- `languages` — массив с языками в порядке предпочтения:

```
console.log(navigator.languages);
// Array [ "ru-RU", "ru", "en-US", "en" ]
```

- `onLine` — режим подключения: `true`, если клиент в настоящее время подключен к сети, и `false`, если отключен:

```
console.log(navigator.onLine);        // true
```

- `cookieEnabled` — режим работы cookies: возвращает `true`, если прием cookies разрешен:

```
console.log(navigator.cookieEnabled); // true
```

Мы уже не раз говорили, что разные Web-браузеры могут по-разному выполнять программный код. К сожалению, пользователь может изменить значения свойств объекта `navigator` в настройках Web-браузера, поэтому не стоит полагаться на эти значения. Вместо определения названия и версии лучше использовать метод проверки функциональных возможностей Web-браузера. Например, проверить наличие необходимого метода, указав его имя без круглых скобок в операторе `if`.

3.14.5. Объект *screen*. Получение информации о мониторе пользователя

Объект `screen` содержит информацию о характеристиках монитора клиента.

Свойства объекта `screen`:

- `width` — полная ширина экрана в пикселах;
- `height` — полная высота экрана в пикселах:

```
console.log(window.screen.width);  // 1920
console.log(window.screen.height); // 1080
```

- `availWidth` — ширина, доступная для окна Web-браузера;

- `availHeight` — высота, доступная для окна Web-браузера:

```
console.log(screen.availWidth);  // 1920
console.log(screen.availHeight); // 1040
```

- `colorDepth` — возвращает глубину цвета:

```
console.log(screen.colorDepth);  // 24
```

3.14.6. Объект *location*. Разбор составляющих URL-адреса документа

Объект `location` содержит информацию об URL-адресе текущей Web-страницы.

Свойства объекта `location`:

- `href` — полный URL-адрес документа (доступно для чтения и записи):

```
console.log(window.location.href);
```

- `protocol` — идентификатор протокола;
- `port` — номер порта;
- `host` — имя хоста, вместе с номером порта;
- `hostname` — имя домена;
- `pathname` — путь и имя файла;
- `search` — строка параметров, указанная после знака `?` (включая этот знак);
- `hash` — имя «якоря» (закладки) в документе, указанное после знака `#` (включая этот знак).

Методы объекта `location`:

- `assign(<URL-адрес>)` — загружает документ, URL-адрес которого указан в качестве параметра;
- `reload([<Флаг>])` — перезагружает документ. Если в параметре указано значение `true`, то страница должна быть перезагружена с сервера, а если `false` (или параметр не указан) — то из кэша;
- `replace(<URL-адрес>)` — загружает документ, URL-адрес которого указан в качестве параметра. При этом информация об URL-адресе предыдущего документа удаляется из объекта `history`.

Загрузить новый документ можно не только с помощью методов `assign()` или `replace()`, но и присвоив новый URL-адрес свойству `href` объекта `location`:

```
window.location.href = "newURL.html";
```

3.14.7. Объект *history*. Получение информации о просмотренных страницах

Объект `history` предоставляет доступ к списку всех просмотренных за последнее время Web-страниц.

Свойство `length` объекта `history` возвращает размер списка истории:

```
console.log(window.history.length);
```

Методы объекта `history`:

- `back()` — загружает в окно предыдущий документ из списка истории;
- `forward()` — загружает в окно следующий документ из списка истории;
- `go(<Позиция>)` — перемещается в списке истории на позицию, относительный номер которой указан в качестве параметра (можно указать как положительное, так и отрицательное значение).

3.14.8. Объект *document*. Получение полной информации о HTML-документе

Объект `document` предоставляет доступ ко всем элементам Web-страницы.

Свойства объекта `document`:

- `documentElement` — ссылка на корневой элемент (тег `<html>`);
- `head` — ссылка на все содержимое тега `<head>`;
- `title` — название документа, указанное в теге `<title>`. Можно получить название и изменить его:

  ```
  window.document.title = "Новое название";
  console.log(document.title); // Новое название
  ```

- `body` — ссылка на все содержимое тега `<body>`;
- `activeElement` — ссылка на активный элемент документа;
- `URL` — URL-адрес документа в виде строки;
- `location` — объект `location` (см. *разд. 3.14.6*):

  ```
  console.log(document.location.href);
  ```

- `referrer` — URL-адрес, с которого перешел посетитель по ссылке, в виде строки;
- `cookie` — объект `cookie`, который позволяет сохранять данные на компьютере клиента;
- `readyState` — статус документа. Возвращает следующие строковые значения:
  - `complete` — документ полностью загружен;
  - `interactive` — Web-страница загружена не полностью, но документ доступен для просмотра и управления;
  - `loading` — документ загружается;
- `lastModified` — дата и время последнего изменения файла документа в виде строки:

  ```
  console.log(document.lastModified);
  ```

Методы объекта `document`:

- `getElementById(<Идентификатор>)` — возвращает ссылку на элемент с указанным идентификатором или значение `null`:

```
var div1 = document.getElementById("div1");
```

- `getElementsByName(<Название элемента>)` — возвращает ссылку на коллекцию элементов, у которых параметр `name` равен значению аргумента;
- `getElementsByTagName(<Тег>)` — возвращает ссылку на коллекцию дочерних элементов, созданных с использованием тега, переданного в качестве параметра;
- `getElementsByClassName(<Стилевой класс>)` — возвращает ссылку на коллекцию элементов, для которых задан указанный стилевой класс (название указывается без точки вначале):

```
<div id="div1" class="cls1"></div>
...
var arr = document.getElementsByClassName("cls1");
console.log(arr[0].id); // div1
```

- `querySelector(<CSS-селектор>)` — возвращает ссылку на первый элемент, соответствующий указанному CSS-селектору, или значение `null`:

```
<div id="div1" class="cls1"></div>
...
console.log(document.querySelector(".cls1").id);        // div1
console.log(document.querySelector("#div1").className); // cls1
```

- `querySelectorAll(<CSS-селектор>)` — возвращает ссылку на коллекцию элементов, соответствующих указанному CSS-селектору.

  Получим содержимое всех абзацев:

```
<p>Абзац 1</p><p>Абзац 2</p>
...
var arr = document.querySelectorAll("p");
for (var i = 0; i < arr.length; i++) {
   console.log(arr[i].innerHTML);
}
```

- `elementFromPoint(<X>, <Y>)` — возвращает ссылку на элемент, находящийся по координатам `<X>` и `<Y>`, или значение `null`;
- `write(<HTML-код>)` — записывает текст, переданный как параметр, в текущее место документа:

```
document.write("<p>Абзац</p>");
```

- `writeln(<HTML-код>)` — записывает текст, переданный как параметр, в текущее место документа, а в конце строки добавляет символы возврата каретки и перевода строки. Методы `write()` и `writeln()` обычно используются при формировании документа на этапе загрузки.

Коллекции объекта `document`:

- `forms` — коллекция всех форм;
- `images` — коллекция всех изображений;
- `links` — коллекция ссылок;
- `scripts` — коллекция скриптов.

Получить число элементов коллекции позволяет свойство `length`. Доступ к элементам коллекции осуществляется по индексу внутри квадратных скобок.

Получим URL-адреса всех ссылок:

```
var arr = document.links;
for (var i = 0; i < arr.length; i++) {
   console.log(arr[i].href);
}
```

3.14.9. Узлы документа

Стандарт DOM предоставляет следующие свойства для получения информации об узле и передвижения по дереву HTML-документа:

- `nodeType` — тип узла. Может принимать следующие значения:
  - 1 — `ELEMENT_NODE` — тег;
  - 3 — `TEXT_NODE` — простой текст;
  - 8 — `COMMENT_NODE` — комментарий;
  - 9 — `DOCUMENT_NODE` — HTML-документ;
- `nodeName` — имя узла. Например, название тега для узла `ELEMENT_NODE`;
- `nodeValue` — значение узла. Например, текст для узлов `TEXT_NODE` и `COMMENT_NODE`;
- `childNodes` — список всех дочерних узлов:

  ```
  var arr = document.body.childNodes;
  for (var i = 0; i < arr.length; i++) {
     console.log(arr[i].nodeType);
     console.log(arr[i].nodeName);
     console.log(arr[i].nodeValue);
  }
  ```

- `firstChild` — первый дочерний узел или значение `null`, если такого узла нет;
- `lastChild` — последний дочерний узел или значение `null`, если узла нет;
- `parentNode` — родительский узел или значение `null`, если такого узла нет;
- `previousSibling` — узел, непосредственно следующий перед указанным узлом, или значение `null`, если такого узла нет;
- `nextSibling` — узел, непосредственно следующий после указанного узла, или значение `null`, если такого узла нет.

Создать новый узел и добавить его в HTML-документ позволяют следующие методы:

- `createElement(<Название тега>)` — создает новый узел `ELEMENT_NODE`;
- `createTextNode(<Текст>)` — создает новый узел `TEXT_NODE`;
- `appendChild(<Новый узел>)` — добавляет новый узел в конец указанного узла. Если узел уже находится в документе, то удаляет его и вставляет в новое место. В качестве примера создадим абзац со ссылкой и добавим его в конец документа:

```
var p = document.createElement("p"); // Создаем абзац
var text = document.createTextNode("Это текст абзаца ");
p.appendChild(text); // Добавляем текст в абзац
var link = document.createElement("a"); // Создаем ссылку
link.href = "test.html"; // Задаем URL-адрес ссылки
var link_text = document.createTextNode("Это текст ссылки");
link.appendChild(link_text); // Добавляем текст в ссылку
p.appendChild(link);         // Добавляем ссылку в конец абзаца
// Добавляем новый узел в конец документа
document.body.appendChild(p);
```

- `insertBefore(<Новый узел>, <Узел>)` — добавляет новый узел перед указанным узлом. Если узел уже находится в документе, то удаляет его и вставляет в новое место:

```
var p = document.getElementsByTagName("p")[0];
// Перемещаем первый абзац в конец документа
document.body.insertBefore(p, document.body.lastChild);
```

  Вставим новый абзац в начало документа:

```
var p = document.createElement("p"); // Создаем абзац
var text = document.createTextNode("Это текст абзаца ");
p.appendChild(text); // Добавляем текст в абзац
// Добавляем новый узел в начало документа
document.body.insertBefore(p, document.body.firstChild);
```

- `cloneNode(true | false)` — создает копию узла. Если в качестве параметра задано значение `true`, то будут скопированы все потомки указанного узла. Обратите внимание: обработчики событий не копируются.

  Сделаем полную копию абзаца и добавим новый узел в конец документа:

```
var p = document.getElementsByTagName("p")[0].cloneNode(true);
// Добавляем в конец документа
document.body.appendChild(p);
```

- `hasChildNodes()` — возвращает значение `true`, если узел имеет дочерние узлы, или `false` — в противном случае:

```
var p = document.getElementsByTagName("p")[0];
if (p.hasChildNodes()) console.log("Есть");
else console.log("Дочерних узлов нет");
```

- removeChild(<Удаляемый узел>) — удаляет узел и возвращает его. В качестве примера удалим первый абзац:

```
var p = document.getElementsByTagName("p")[0];
p.parentNode.removeChild(p);
```

- replaceChild(<Новый узел>, <Старый узел>) — заменяет узел другим узлом. Заменим первый тег <div> на новый абзац:

```
var p = document.createElement("p"); // Создаем абзац
var text = document.createTextNode("Это текст абзаца");
p.appendChild(text); // Добавляем текст в абзац
var div = document.getElementsByTagName("div")[0];
div.parentNode.replaceChild(p, div);
```

Для получения информации об элементах таблицы предназначены специальные свойства:

- caption — ссылка на элемент CAPTION или значение null, если он не существует;
- tHead — ссылка на элемент THEAD или значение null, если он не существует;
- tFoot — ссылка на элемент TFOOT или значение null, если он не существует;
- tBodies — коллекция всех элементов TBODY в таблице;
- rows — коллекция всех строк в таблице или в секции TBODY при использовании свойства tBodies. Каждый элемент коллекции содержит следующие свойства:
  - cells — коллекция всех ячеек в строке таблицы. Каждый элемент коллекции содержит свойство cellIndex, через которое доступен индекс ячейки в строке;
  - rowIndex — индекс строки в таблице;
  - sectionRowIndex — индекс строки внутри раздела (THEAD, TBODY или TFOOT).

Для создания и удаления элементов таблицы предназначены следующие методы:

- createCaption() — создает новый элемент CAPTION или возвращает ссылку на существующий элемент.

  Добавим заголовок к таблице:

```
var table = document.getElementsByTagName("table")[0];
var caption = table.createCaption();
var text = document.createTextNode("Это заголовок таблицы");
caption.appendChild(text); // Добавляем текст в элемент CAPTION
```

- deleteCaption() — удаляет элемент CAPTION;
- createTHead() — создает новый элемент THEAD или возвращает ссылку на существующий элемент;
- deleteTHead() — удаляет элемент THEAD;
- createTFoot() — создает новый элемент TFOOT или возвращает ссылку на существующий элемент;
- deleteTFoot() — удаляет элемент TFOOT;

- `deleteRow(<Индекс>)` — удаляет строку из таблицы по указанному индексу;
- `insertRow(<Индекс>)` — вставляет новый пустой элемент `TR` в указанную позицию;
- `insertCell(<Индекс>)` — вставляет пустой элемент `TD`.

  Добавим новый ряд в самое начало первой таблицы в HTML-документе:

```
var table = document.getElementsByTagName("table")[0];
var row = table.insertRow(0);
var cell1 = row.insertCell(0);
var cell2 = row.insertCell(1);
var t1 = document.createTextNode("Первая ячейка");
var t2 = document.createTextNode("Вторая ячейка");
cell1.appendChild(t1);
cell2.appendChild(t2);
```

- `deleteCell(<Индекс>)` — удаляет указанную ячейку.

3.14.10. Общие свойства и методы элементов Web-страницы

Все элементы Web-страницы также имеют свойства и методы. Помимо свойств, специфических для конкретных элементов, все они имеют следующие общие свойства:

- `id` — уникальный идентификатор элемента, заданный параметром `id`;
- `className` — имя стилевого класса, заданное параметром `class`;
- `tagName` — имя тега элемента:

```
<div id="div1" class="cls1">Текст</div>
...
var elem = document.getElementById('div1');
console.log(elem.id);        // div1
console.log(elem.className); // cls1
console.log(elem.tagName);   // DIV
```

- `parentElement` — ссылка на элемент-родитель;
- `innerHTML` — содержимое элемента, включая внутренние теги HTML. Если присвоить свойству новое значение, то содержимое элемента также изменится:

```
var elem = document.getElementById('div1');
elem.innerHTML = "<b>Новый текст</b>";
```

  Результат в HTML-коде:

```
<div id="div1"><b>Новый текст</b></div>
```

- `textContent` — содержимое элемента в виде текста. Если присвоить свойству новое значение, то содержимое элемента также изменится:

```
var elem = document.getElementById('div1');
elem.textContent = "Новый текст";
console.log(elem.textContent); // Новый текст
```

- `attributes` — возвращает коллекцию всех параметров тега. Каждый элемент коллекции содержит два свойства:
  - `name` — название параметра;
  - `value` — значение параметра.

  Пример:

```
var elem = document.getElementById('div1');
var arr = elem.attributes;
for (var i = 0; i < arr.length; i++) {
   console.log(arr[i].name + " = " + arr[i].value);
}
```

Общие методы элементов Web-страницы:

- `getAttribute(<Имя параметра>)` — возвращает значение параметра с именем `<Имя параметра>` тега текущего элемента:

```
var elem = document.getElementById('div1');
console.log(elem.getAttribute("id")); // div1
```

- `setAttribute(<Имя параметра>, <Значение>)` — присваивает `<Значение>` параметру с именем `<Имя параметра>` тега текущего элемента;
- `removeAttribute(<Имя параметра>)` — удаляет параметр тега текущего элемента;
- `hasAttribute(<Имя параметра>)` — возвращает значение `true`, если параметр с указанным именем существует, и `false` — в противном случае;
- `getElementsByTagName(<Тег>)` — возвращает ссылку на коллекцию дочерних элементов, созданных с использованием тега, переданного в качестве параметра;
- `getElementsByClassName(<Стилевой класс>)` — возвращает ссылку на коллекцию дочерних элементов, для которых задан указанный стилевой класс (название указывается без точки в начале);
- `querySelector(<CSS-селектор>)` — возвращает ссылку на первый дочерний элемент, соответствующий указанному CSS-селектору, или значение `null`;
- `querySelectorAll(<CSS-селектор>)` — возвращает ссылку на коллекцию дочерних элементов, соответствующих указанному CSS-селектору;
- `scrollIntoView(true | false)` — вызывает прокрутку страницы в окне так, чтобы текущий элемент оказался в поле зрения. Если параметр равен `true`, то текущий элемент окажется у верхнего края окна, а если `false` — то у нижнего края.

Мы можем обращаться к параметрам любого тега напрямую:

- изменим адрес гиперссылки:

```
<a href="doc1.html" id="link1">Текст ссылки</a>
<script>
```

```
document.getElementById("link1").href = "doc2.html";
</script>
```

- изменим адрес изображения:

```
<img src="image1.gif" id="image1">
<script>
document.getElementById("image1").src = "image2.gif";
</script>
```

Какие параметры имеет тот или иной тег и какие значения он может принимать, мы рассматривали при изучении языка HTML в *главе 1*.

3.14.11. Работа с таблицами стилей при помощи JavaScript

Свойство `style` позволяет получить доступ к стилям элемента, указанным в параметре `style`. Для того чтобы получить значение атрибута стиля, заданное вне тега, нужно использовать не свойство `style`, а метод `getComputedStyle()` объекта `window`:

```
var elem = document.getElementById('div1');
console.log(window.getComputedStyle(elem, null).width); // 956px
```

Обратите внимание: метод `getComputedStyle()` возвращает вычисленное значение в виде строки, а не то значение, которое было указано в CSS. Если значение атрибута было указано в процентах, то мы получим в большинстве случаев абсолютное значение на момент вызова метода.

Свойства объекта `style` соответствуют атрибутам в каскадных таблицах стилей с небольшими отличиями в написании:

- символы дефиса (-) удаляются;
- первые буквы всех слов в названии атрибута, кроме первого, делаются прописными.

Приведем примеры преобразования атрибутов стиля в свойства объекта `style`:

```
color -> color
font-family -> fontFamily
font-size -> fontSize
border-left-style -> borderLeftStyle
```

Пример указания значений:

```
<div id="div1">Текст</div>
...
var elem = document.getElementById('div1');
elem.style.color = "red";
elem.style.fontFamily = "Verdana";
elem.style.fontSize = "14pt";
elem.style.borderLeftStyle = "solid";
```

Атрибуты стилей и их допустимые значения мы рассматривали при изучении CSS в *главе 2*. По аналогии с приведенными примерами можно преобразовать названия атрибутов стиля в свойства объекта `style`.

3.14.12. Объект *selection*. Проверка наличия выделенного фрагмента

Объект `selection` позволяет получить доступ к тексту, выделенному в окне Web-браузера. Для получения выделенного текста в документе (но не в текстовых полях) применяется метод `getSelection()` объекта `window`. Объект, возвращаемый методом `getSelection()`, содержит следующие свойства:

- `anchorNode` — возвращает ссылку на текстовый узел, в котором началось выделение или значение `null`.

  Получим элемент, в котором началось выделение, и сделаем его фон красным:

  ```
  var rng = window.getSelection();
  if (rng.anchorNode)
      rng.anchorNode.parentNode.style.backgroundColor = "red";
  ```

- `anchorOffset` — возвращает смещение от начала текстового узла (возвращаемого свойством `anchorNode`) до начальной границы выделения;
- `focusNode` — возвращает ссылку на текстовый узел, в котором закончилось выделение;
- `focusOffset` — возвращает смещение от начала текстового узла (возвращаемого свойством `focusNode`) до конечной границы выделения:

  ```
  var rng = window.getSelection();
  if (rng.focusNode) {
     rng.focusNode.parentNode.style.backgroundColor = "red";
     console.log(rng.focusOffset);
  }
  ```

- `rangeCount` — возвращает количество объектов `Range`, которые входят в выделенный фрагмент;
- `isCollapsed` — возвращает `true`, если объект свернут в точку, и `false` — в противном случае.

Методы объекта `selection`:

- `toString()` — возвращает текстовое содержимое выделенного фрагмента:

  ```
  var rng = window.getSelection();
  if (!rng.isCollapsed) {
     console.log(rng.toString());
  }
  ```

- `collapse(<Узел>, <Смещение>)` — сворачивает выделение в указанную точку;

- `collapseToStart()` — сворачивает выделение в начало фрагмента;
- `collapseToEnd()` — сворачивает выделение в конец фрагмента;
- `deleteFromDocument()` — удаляет выделенный фрагмент из документа;
- `extend(<Узел>, <Смещение>)` — перемещает конечную границу выделенного фрагмента в указанную позицию.

  В качестве примера расширим (или уменьшим в зависимости от направления выделения) фрагмент до конца узла, содержащего конечную границу:

  ```
  var rng = window.getSelection();
  if (!rng.isCollapsed) {
     var len = rng.focusNode.length;
     if (rng.focusOffset!=len) {
        rng.extend(rng.focusNode, len);
     }
  }
  else window.alert("Нет выделенного фрагмента");
  ```

- `getRangeAt(<Индекс>)` — возвращает объект `Range` по указанному индексу;
- `addRange(<Объект Range>)` — добавляет указанный объект `Range` к текущему выделению;
- `removeRange(<Объект Range>)` — удаляет указанный объект `Range` из выделенного фрагмента;
- `removeAllRanges()` — удаляет все объекты `Range` из выделенного фрагмента;
- `selectAllChildren(<Узел>)` — выделяет текстовое содержимое указанного узла и всех его потомков.

  Выделим содержимое первого абзаца в документе:

  ```
  var rng = window.getSelection();
  var elem = document.getElementsByTagName("p")[0];
  rng.selectAllChildren(elem);
  ```

Фрагмент, выделенный в текстовом поле, нельзя получить с помощью метода `getSelection()`. Вместо этого метода следует использовать свойства `selectionStart` и `selectionEnd`:

```
<input type="text" name="txt1" id="txt1">
...
// Ссылка на текстовое поле
var elem = document.getElementById("txt1");
if (elem.selectionStart != undefined &&
    elem.selectionEnd != undefined) {
   var s = elem.selectionStart; // Начальная позиция
   var e = elem.selectionEnd;   // Конечная позиция
   window.alert(elem.value.substring(s, e));
}
```

Выделить фрагмент внутри текстового поля позволяет метод `setSelectionRange(<Начало>, <Конец>)`:

```
// Ссылка на текстовое поле
var elem = document.getElementById("txt1");
elem.focus();
elem.setSelectionRange(0, 5);
```

3.14.13. Объект *Range*. Расширение или сжатие выделенного фрагмента текста

Объект `Range` предоставляет доступ к фрагменту текста Web-страницы. Создать область позволяет метод `createRange()` объекта `document`:

```
var rng = document.createRange();
```

Можно также получить ссылку на выделенную область с помощью метода `getRangeAt(<Индекс>)` объекта `selection`:

```
var sel = window.getSelection();
if (!sel.isCollapsed) {
   var rng = sel.getRangeAt(0);
}
```

Свойства объекта `Range`:

- `startContainer` — возвращает ссылку на узел, в котором содержится начальная точка области;
- `startOffset` — возвращает смещение от начала узла (возвращаемого свойством `startContainer`) до начальной точки области;
- `endContainer` — возвращает ссылку на узел, в котором содержится конечная точка области;
- `endOffset` — возвращает смещение от начала узла (возвращаемого свойством `endContainer`) до конечной точки области:

```
<p id="txt1">Текст внутри абзаца</p>
...
var elem = document.getElementById("txt1").firstChild;
var rng = document.createRange();
rng.setStart(elem, 1);      // Начальная точка
rng.setEnd(elem, 5);        // Конечная точка
console.log(rng.startContainer);
console.log(rng.startOffset);
console.log(rng.endContainer);
console.log(rng.endOffset);
var sel = window.getSelection();
sel.addRange(rng);          // Выделяем область
```

- `collapsed` — возвращает `true`, если объект свернут в точку, и `false` — в противном случае;

- commonAncestorContainer — возвращает ссылку на узел, в котором содержатся как начальная, так и конечная точки области.

Методы объекта Range:

- toString() — возвращает текстовое содержимое области;
- setStart(<Узел>, <Смещение>) — устанавливает положение начальной точки области;
- setEnd(<Узел>, <Смещение>) — устанавливает положение конечной точки области;
- setStartBefore(<Узел>) — устанавливает начальную точку области перед указанным узлом;
- setStartAfter(<Узел>) — устанавливает начальную точку области после указанного узла;
- setEndBefore(<Узел>) — устанавливает конечную точку области перед указанным узлом;
- setEndAfter(<Узел>) — устанавливает конечную точку области после указанного узла;
- cloneRange() — создает копию объекта Range;
- cloneContents() — создает копию внутреннего содержимого области. В качестве значения возвращает объект DocumentFragment;
- detach() — удаляет объект Range;
- deleteContents() — удаляет все внутреннее содержимое области из документа;
- extractContents() — удаляет все внутреннее содержимое области из документа и возвращает объект DocumentFragment, в котором будет находиться удаленное содержимое области;
- collapse(<true | false>) — сворачивает область в указанную точку. Если в качестве параметра указано значение true, то область сворачивается в начальную точку, а если false — то в конечную точку;
- selectNode(<Узел>) — ограничивает область указанным в качестве параметра узлом;
- selectNodeContents(<Узел>) — ограничивает область внутренним содержимым указанного узла;
- insertNode(<Узел>) — вставляет новый узел в начало области;
- surroundContents(<Узел>) — вкладывает содержимое области в указанный узел;
- compareBoundaryPoints(<Точки сравнения>, <Область, с которой сравниваем>) — сравнивает позиции двух областей. В качестве первого параметра могут быть указаны следующие значения:
  - 0 — START_TO_START — сравнение начальных точек;
  - 1 — START_TO_END — сравнение начальной точки области, указанной в качестве второго параметра, с конечной точкой данной области;

- 2 — END_TO_END — сравнение конечных точек;
- 3 — END_TO_START — сравнение конечной точки области, указанной в качестве второго параметра, с начальной точкой данной области.

В качестве примера использования объекта Range найдем внутри абзаца текст "фрагмент" и вложим его в тег <strong>:

```
<p id="txt">Текст для выделения фрагмента</p>
<input type="button" id="btn1" onclick="handler()" value="Выделить">
<script>
function handler() {
   if (document.createRange) {
      var p = document.getElementById("txt").firstChild;
      var text = p.nodeValue; // Получаем текст абзаца
      var ind = text.indexOf("фрагмент");
      if (ind != -1) { // Если текст найден
         // Создаем объект Range
         var rng = document.createRange();
         rng.setStart(p, ind); // Начальная точка
         // Конечная точка
         rng.setEnd(p, ind + 8);
         // Элемент, в который будем вкладывать текст
         var s = document.createElement("strong");
         // Вкладываем область в тег strong
         rng.surroundContents(s);
      }
   }
   else {
      window.alert("Web-браузер не поддерживает метод createRange");
   }
}
</script>
```

Теперь изменим цвет фона текстового фрагмента, выделенного пользователем:

```
<p id="txt">Текст для выделения фрагмента</p>
<input type="button" id="btn1" onclick="handler()" value="Выделить">
<script>
function handler() {
   if (document.createRange && window.getSelection) {
      var sel = window.getSelection();
      if (!sel.isCollapsed) {
         var rng = sel.getRangeAt(0);
         sel.collapseToStart(); // Убираем выделение
         // Элемент, в который будем вкладывать выделенный текст
         var s = document.createElement("span");
         s.style.backgroundColor = "#FFE9B3";
         // Вкладываем область в тег span
         rng.surroundContents(s);
      }
```

```
      else window.alert("Нет выделенного фрагмента");
   }
   else {
      window.alert("Web-браузер не поддерживает методы");
   }
}
</script>
```

3.14.14. Сохранение данных на компьютере клиента

Web-браузеры позволяют сохранять небольшой объем информации в специальном текстовом файле на компьютере пользователя. Такие файлы называются *cookies*. Возможность использования cookies можно отключить в настройках Web-браузера. Для проверки возможности использования cookies следует задействовать свойство `cookieEnabled` объекта `navigator`:

```
if (navigator.cookieEnabled) {
   window.alert("Использование cookies разрешено");
}
```

Запись cookies производится путем присвоения значения свойству `cookie` объекта `document` в следующем формате:

```
document.cookie = "<Имя>=<Значение>; [expires=<Дата>;]
                   [domain=<Имя домена>;] [path=<Путь>;] [secure;]";
```

Здесь используются следующие параметры:

- `<Имя>=<Значение>` — задает имя сохраняемой переменной и ее значение. Это единственный обязательный параметр. Если не задан параметр `expires`, то по истечении текущего сеанса работы Web-браузера cookies будут автоматически удалены;
- `expires` — указывает дату удаления cookies в следующем формате:

  ```
  Thu, 01 Jan 1970 00:00:01 GMT
  ```

  Получить дату в этом формате можно с помощью методов `setTime()` и `toUTCString()` класса `Date`. Методу `setTime()` нужно передать текущее время в миллисекундах плюс время хранения cookies в миллисекундах. Текущее время можно получить с помощью метода `getTime()`. Рассчитать время хранения cookies можно исходя из следующих соотношений:

  - 1 секунда = 1 000 миллисекунд;
  - 1 минута = 60 секунд = 60 000 миллисекунд;
  - 1 час = 60 минут = 3 600 секунд = 3 600 000 миллисекунд;
  - 1 день = 24 часа = (24×3 600 000) миллисекунд = 86 400 000 миллисекунд.

  Например:

  ```
  var d = new Date();
  d.setTime(d.getTime() + 3600000); // Задан 1 час
  var endDate = d.toUTCString();    // Дата удаления cookies
  ```

- domain=<Имя домена> — задает доменную часть URL-адреса, для которой действует созданный файл cookies;
- path=<Путь> — задает часть URL-адреса, определяющую путь к документам, для которых действует созданный файл cookies.

Считывание cookies производится с помощью свойства cookie объекта document:

```
var cookies = document.cookie;
```

Переменная cookies будет содержать строку, в которой перечислены все установленные пары имя=значение через точку с запятой:

```
"имя1=значение1; имя2=значение2"
```

Для удаления cookies следует установить cookies с прошедшей датой.

В качестве примера сохраним имя и фамилию пользователя, и при следующем посещении будем приветствовать его, используя сохраненные данные (листинг 3.60). Добавим также возможность удаления cookies. Для совместимости закодируем введенные данные с помощью метода encodeURIComponent(), а при выводе раскодируем их с помощью метода decodeURIComponent(). Это позволяет безопасно сохранять значения, введенные кириллицей.

Листинг 3.60. Установка и удаление cookies

```
<!DOCTYPE html>
<html lang="ru">
<head>
 <meta charset="utf-8">
 <title>Установка и удаление cookies</title>
<script>
function saveCookies() {
   if (navigator.cookieEnabled) {
      var txt1 = document.getElementById("txt1");
      var txt2 = document.getElementById("txt2");
      if (txt1.value != "" && txt2.value != "") {
         var d = new Date();
         d.setTime(d.getTime() + 3600000); // Задан 1 час
         var endDate = d.toUTCString();    // Дата удаления cookies
         var str = "name1=" + encodeURIComponent(txt1.value);
         str += ";expires=" + endDate + ";";
         document.cookie = str;
         str = "name2=" + encodeURIComponent(txt2.value);
         str += ";expires=" + endDate + ";";
         document.cookie = str;
         txt1.value = "";
         txt2.value = "";
         showCookies();
      }
```

```
      else {
         window.alert("Не заполнено обязательное поле");
      }
   }
}
function deleteCookies() {
   if (navigator.cookieEnabled) {
      if (document.cookie != "") {
         var d = new Date();
         d.setTime(1000); // Дата в прошлом
         var endDate = d.toUTCString();
         document.cookie = "name1=;expires=" + endDate + ";";
         document.cookie = "name2=;expires=" + endDate + ";";
         showCookies();
      }
   }
}
function showCookies() {
   if (navigator.cookieEnabled) {
      var div1 = document.getElementById("div1");
      if (document.cookie != "") {
         var arr1 = [], arr2 = [];
         var obj = {};
         var str = document.cookie;
         if (str.indexOf("; ") != -1) {
            arr1 = str.split("; ");
            for (var i = 0, c = arr1.length; i < c; i++) {
               arr2 = arr1[i].split("=");
               obj[arr2[0]] = arr2[1];
            }
         }
         else {
            arr2 = str.split("=");
            obj[arr2[0]] = arr2[1];
         }
         var msg = "Привет, ";
         msg += decodeURIComponent(obj.name2).replace("<", "&lt;");
         msg += " " + decodeURIComponent(obj.name1).replace("<", "&lt;");
         div1.innerHTML = msg;
      }
      else div1.innerHTML = "";
   }
}
window.onload = function() {
   showCookies();
}
</script>
</head>
```

```
<body>
<div id="div1"></div>
<div>
Введите ваше имя:<br>
<input type="text" id="txt1"><br>
Введите вашу фамилию:<br>
<input type="text" id="txt2"><br>
<input type="button" value="Сохранить" onclick="saveCookies()"><br>
<input type="button" value="Удалить cookies" onclick="deleteCookies()">
</div>
</body>
</html>
```

3.15. Работа с элементами формы

При изучении HTML мы рассмотрели создание элементов форм. В этом разделе мы научимся с помощью JavaScript обрабатывать данные, введенные пользователем в элементы формы. Обработка на стороне клиента позволит снизить нагрузку на Web-сервер за счет отмены отправки данных формы при неправильно введенных значениях.

3.15.1. Элементы управления

Для начала еще раз приведем основные элементы форм:

- `<input type="text">` — текстовое поле ввода;
- `<input type="password">` — текстовое поле для ввода пароля;
- `<input type="file">` — позволяет отправить файл на Web-сервер;
- `<input type="checkbox">` — поле для установки флажка;
- `<input type="radio">` — элемент-переключатель;
- `<input type="reset">` — кнопка, при нажатии которой вся форма очищается;
- `<input type="submit">` — кнопка, при нажатии которой происходит отправка данных на Web-сервер;
- `<input type="button">` — обычная командная кнопка;
- `<input type="hidden">` — скрытый элемент формы;
- `<textarea>Текст</textarea>` — поле для ввода многострочного текста;
- `<select><option>Элемент</option></select>` — список с возможными значениями.

В HTML 5 были добавлены элементы: `url`, `email`, `tel`, `number`, `range`, `color`, `search`, `date`, `time`, `datetime-local`, `month` и `week` (см. *разд. 1.11.4*).

Все элементы должны быть расположены внутри тегов `<form>` и `</form>`. Именно форма определяет, что делать с данными дальше. Параметр `action` задает URL-ад-

рес программы обработки формы, параметр `method` определяет, как будут пересылаться данные от формы до Web-сервера (методом `GET` или `POST`), а параметр `enctype` задает MIME-тип передаваемых данных. С помощью параметра `name` задается уникальное имя формы, благодаря которому можно управлять элементами формы из скриптов.

Параметр `name` необходимо указывать во всех элементах формы, за исключением кнопок. Именно имя элемента, заданное в параметре `name`, пересылается на Web-сервер вместе со значением элемента формы. Имя элемента в пределах формы должно быть уникальным, за исключением переключателей, объединенных в группу.

Для доступа к элементам формы из скриптов необходимо указать параметр `id`. Обычно для элементов форм значения параметров `name` и `id` содержат одно и то же имя:

```
<input type="text" name="text1" id="text1">
```

Если данные не нужно отправлять на Web-сервер, то можно вообще не использовать тег `<form>`. В этом случае вся обработка осуществляется с помощью скриптов.

3.15.2. Коллекция *forms*. Доступ к элементу формы из скрипта

Все формы документа доступны через коллекцию `forms`. Например, чтобы получить значение текстового поля с именем `text1` (входящего в состав формы `form1`), можно воспользоваться следующей строкой кода:

```
document.forms["form1"].text1.value
```

К отдельной форме можно также обратиться по индексу:

```
document.forms[0].text1.value
```

Получить доступ к элементу вне зависимости от того, находится он внутри формы или нет, позволяет метод `getElementById()` объекта `document`:

```
document.getElementById("text1").value
```

Все элементы формы доступны через коллекцию `elements`:

```
document.forms["form1"].elements["text1"].value
document.forms["form1"].elements[0].value
document.forms[0].elements[0].value
```

3.15.3. Свойства объекта формы

Объект формы поддерживает следующие свойства:

- `length` — количество элементов формы;
- `action` — URL-адрес программы обработки формы;
- `elements` — ссылка на коллекцию `elements`;
- `encoding` — MIME-тип передаваемых данных;

- `method` — режим пересылки данных формы на Web-сервер;
- `enctype` — метод кодирования данных формы;
- `name` — имя формы;
- `target` — имя фрейма, в который будет загружен документ, являющийся результатом обработки данных формы Web-сервером.

В HTML 5 доступны также два новых свойства:

- `autocomplete` — признак, указывающий, работает ли в форме автодополнение. Если в качестве признака задана строка `"on"`, автодополнение работает, если `"off"` — не работает;
- `noValidate` — признак того, будет ли форма проверять введенные в нее данные на корректность. Значение `true` активизирует проверку данных, значение `false` отключает ее.

3.15.4. Методы объекта формы

Объект формы поддерживает следующие методы:

- `submit()` — выполняет отправку данных формы серверной программе. Аналогично нажатию кнопки **Submit**;
- `reset()` — очищает форму, т. е. все элементы формы получают значения по умолчанию. Аналогично нажатию кнопки **Reset**.

Также нас может заинтересовать метод `checkValidity()`. Он возвращает `true`, если во все элементы управления формы занесены корректные данные, и `false` — в противном случае:

```
<form action="test.php" id="frm">
<input type="url" name="url" required>
<input type="button" value="Проверить" onclick="handler()">
</form>
<script>
function handler() {
   var frm = document.getElementById("frm");
   if (!frm.checkValidity()) {
      window.alert("Введите корректные данные!");
   }
}
</script>
```

3.15.5. События объекта формы

Объект формы поддерживает следующие события:

- `onsubmit` — наступает при отправке данных формы;
- `onreset` — возникает при очистке формы.

Элементы управления имеют свои свойства, методы и события. Рассмотрим каждый тип элементов формы по отдельности.

3.15.6. Текстовое поле и поле ввода пароля

Текстовое поле и поле для ввода пароля имеют одинаковые свойства:

- `value` — значение элемента формы;
- `defaultValue` — начальное значение, заданное параметром `value`;
- `disabled` — запрет элемента формы: если задано значение `true`, то поле является неактивным (отображается серым цветом);
- `form` — ссылка на форму, в которой находится элемент;
- `maxLength` — максимальное количество символов, которое может быть введено в поле;
- `name` — имя элемента;
- `type` — тип элемента формы;
- `readOnly` — запрет редактирования: если задано значение `true`, текст в поле нельзя редактировать, если `false` — можно.

Методы тоже одинаковы:

- `blur()` — убирает фокус ввода с текущего элемента формы;
- `focus()` — помещает фокус на текущий элемент формы;
- `select()` — выделяет текст в поле.

Обоими элементами поддерживаются следующие события:

- `onblur` — происходит при потере фокуса элементом формы;
- `onchange` — наступает после изменения данных в поле и при переводе фокуса ввода на другой элемент либо при отправке данных формы. Наступает перед событием `onblur`;
- `onfocus` — возникает при получении фокуса ввода элементом формы.

Кроме указанных событий можно использовать стандартные события мыши и клавиатуры (см. *разд. 3.13.9* и *3.13.10*).

В качестве примера рассмотрим форму ввода E-mail и пароля с проверкой правильности ввода (листинг 3.61). Если данные введены неправильно, то при отправке формы:

- поле выделяется розовым цветом;
- текст в поле выделяется;
- выводится сообщение об ошибке;
- отправка формы прерывается.

Поле **Повтор E-mail** запрещено для редактирования. При вводе адреса электронной почты данные автоматически копируются из поля E-mail в поле **Повтор E-mail**.

Листинг 3.61. Форма ввода E-mail и пароля с проверкой правильности ввода

```
<!DOCTYPE html>
<html lang="ru">
<head>
 <meta charset="utf-8">
 <title>Пример использования текстовых полей</title>
<script>
function frmSubmit() {
   var email = document.getElementById("email");
   var passwd = document.getElementById("passwd");
   email.style.backgroundColor = "#FFFFFF";
   passwd.style.backgroundColor = "#FFFFFF";
   var p = /^[a-z0-9_.-]+@([a-z0-9-]+\.)+[a-z]{2,6}$/i;
   var str = email.value;
   if (!p.test(str)) {
      window.alert("Неверный адрес E-mail");
      email.style.backgroundColor = "#FFE4E1";
      email.select();
      return false;
   }
   p = /^[a-z0-9_.-]{6,16}$/i;
   str = passwd.value;
   if (!p.test(str)) {
      window.alert("Неверный пароль");
      passwd.style.backgroundColor = "#FFE4E1";
      passwd.select();
      return false;
   }
   var msg = "Вы ввели следующие данные:\n\n E-mail: ";
   msg += email.value + "\n Пароль: " + passwd.value;
   window.alert(msg);
   return true;
}
function frmReset() {
   document.getElementById("email").style.backgroundColor = "#FFFFFF";
   document.getElementById("passwd").style.backgroundColor = "#FFFFFF";
}
window.onload = function() {
   document.getElementById("email2").readOnly = true;
   document.getElementById("email").onkeyup = function() {
      document.getElementById("email2").value =
           document.getElementById("email").value;
   }
}
</script>
</head>
```

```
<body>
 <form action="test.php" method="GET" name="frm" id="frm"
       onsubmit="return frmSubmit()" onreset="frmReset()">
  <div>
  E-mail:<br>
  <input type="text" name="email" id="email"
         style="background-color: #FFFFFF"><br>
  Повтор E-mail:<br>
  <input type="text" name="email2" id="email2"
         style="background-color: #FFFFFF"><br>
  Пароль:<br>
  <input type="password" name="passwd" id="passwd"
         style="background-color: #FFFFFF"><br>
  <input type="reset" value="Очистить">
  <input type="submit" value="Отправить">
  </div>
 </form>
</body>
</html>
```

3.15.7. Поле для ввода многострочного текста

Поле для ввода многострочного текста, определяемое парным тегом `<textarea>`, поддерживает те же свойства, методы и события, что и простое поле ввода (см. *разд. 3.15.6*), за исключением свойства `maxLength`. Кроме того, поддерживается еще одно свойство: `wrap` — режим переноса слов. Оно может принимать следующие значения:

- `off` — не переносить слова;
- `physical` — слова переносятся как на экране, так и при передаче данных серверу;
- `virtual` — слова переносятся только на экране, но не при передаче данных серверу.

Для примера рассмотрим возможность добавления слов из текстового поля в поле для ввода многострочного текста (листинг 3.62). Добавить слово можно с помощью кнопки **Добавить слово** или нажатием клавиши <Enter>. Так как по умолчанию нажатие клавиши <Enter> приводит к отправке данных формы, то всплывание события прерывается. При нажатии кнопки **Значение поля** выводится текущее значение тега `<textarea>`.

Листинг 3.62. Добавление слов из текстового поля в поле `<textarea>`

```
<!DOCTYPE html>
<html lang="ru">
<head>
 <meta charset="utf-8">
```

```
 <title>Пример использования поля &lt;TEXTAREA&gt;</title>
<script>
function frmSubmit() {
   var v = document.getElementById("txt1").value;
   window.alert("Текущее значение: \n" + v);
   return false;
}
function btnClick() {
   var txt2 = document.getElementById("txt2");
   var text = txt2.value;
   if (text != "") {
      document.getElementById("txt1").value += text + "\n";
      txt2.value = "";
      txt2.focus();
   }
   else {
      window.alert("Поле не заполнено!");
      txt2.focus();
   }
}
window.onload = function() {
   document.getElementById("txt2").onkeypress = function(e) {
      e = e || window.event;
      if (e.keyCode == 13) {
         btnClick();
         if (e.preventDefault) e.preventDefault();
         else e.returnValue = false;
      }
   }
}
</script>
</head>
<body>
 <form action="test.php" method="GET" name="frm" id="frm"
       onsubmit="return frmSubmit()">
 <div>
  Слово:<br>
  <input type="text" name="txt2" id="txt2"><br>
  <textarea name="txt1" id="txt1" cols="15" rows="10"></textarea>
  <br><input type="button" value="Добавить слово"
             onclick="return btnClick()"><br>
  <input type="submit" value=" Значение поля ">
 </div>
 </form>
</body>
</html>
```

3.15.8. Список с возможными значениями

Объекту списка присущи следующие свойства:

- `disabled` — запрет доступа: если задано значение `true`, то список является неактивным (отображается серым цветом);
- `form` — ссылка на форму, в которой находится элемент;
- `length` — количество пунктов в списке (доступно и для записи);
- `multiple` — разрешение множественного выделения: `true`, если из списка можно выбрать сразу несколько элементов одновременно;
- `name` — имя элемента;
- `options` — ссылка на коллекцию пунктов в списке;
- `selectedOptions` — ссылка на коллекцию выбранных пунктов списка;
- `selectedIndex` — номер выбранного пункта (нумерация начинается с нуля);
- `size` — число одновременно видимых элементов списка;
- `type` — тип элемента формы (`select-multiple` или `select-one`);
- `value` — значение пункта, выбранного в списке.

Пункту списка присущи следующие свойства:

- `defaultSelected` — пункт списка, выбранный изначально;
- `index` — номер пункта в списке;
- `selected` — признак выделения: `true`, если пункт выбран в списке;
- `disabled` — если задано значение `true`, то пункт списка является неактивным (отображается серым цветом);
- `text` — текст пункта списка;
- `value` — значение пункта, выбранного в списке.

Методы:

- `blur()` — убирает фокус ввода с текущего элемента формы;
- `focus()` — помещает фокус на текущий элемент формы.

События:

- `onblur` — наступает при потере фокуса элементом формы;
- `onchange` — происходит после выбора нового пункта списка;
- `onfocus` — наступает при получении фокуса ввода элементом формы.

Рассмотрим пример работы со списками. Документ, приведенный в листинге 3.63, демонстрирует следующие возможности:

- добавление нового пункта списка. При заполнении первого поля и нажатии клавиши <Enter> фокус ввода перемещается во второе поле. При заполнении второго поля и нажатии клавиши <Enter> введенные значения добавляются в список. Вместо клавиши <Enter> можно воспользоваться кнопкой **Добавить**;

- получение всех выбранных значений из списка с возможностью множественного выбора;
- применение взаимосвязанных списков и получение значения выбранного пункта. При выборе элемента в первом списке соответствующие элементы загружаются во второй список. При выборе элемента во втором списке выводится сообщение со значением выбранного пункта;
- применение списков вместо гиперссылок. При выборе элемента списка загружается Web-страница, находящаяся по указанному в параметре `value` URL-адресу.

Листинг 3.63. Обработка списков

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Пример обработки списков</title>
</head>
<body>
<form action="test.php" method="GET" name="frm" id="frm">
<!-- Добавление пункта в список  -->

<script>
function addOption() {
   var txt1 = document.getElementById("txt1");
   var txt2 = document.getElementById("txt2");
   var select1 = document.getElementById("select1");
   if (txt1.value != "" && txt2.value != "") {
      var i = select1.length++;
      select1.options[i].text = txt1.value;
      select1.options[i].value = txt2.value;
      txt1.value = "";
      txt2.value = "";
      txt1.focus();
   }
   else {
      window.alert("Поле не заполнено!");
      txt1.focus();
   }
}
function press1(e) {
   e = e || window.event;
   if (e.keyCode == 13) {
      document.getElementById("txt2").focus();
      if (e.preventDefault) e.preventDefault();
      else e.returnValue = false;
   }
}
```

```
function press2(e) {
   e = e || window.event;
   if (e.keyCode == 13) {
      addOption();
      if (e.preventDefault) e.preventDefault();
      else e.returnValue = false;
   }
}
</script>
<div>
<b>Добавление пункта в список:</b><br><br>
Текст пункта:<br>
<input type="text" name="txt1" id="txt1" onkeypress="press1(event)">
<br>Значение пункта:<br>
<input type="text" name="txt2" id="txt2" onkeypress="press2(event)">
<br><select name="select1" id="select1">
</select><br>
<input type="button" value="Добавить" onclick="addOption()"><br><br>

<!-- Список со множественным выбором  -->

<script>
function multi() {
   var msg = "";
   var select2 = document.getElementById("select2");
   if (select2.selectedOptions) {
      var obj = select2.selectedOptions;
      var count = obj.length;
      for (var i = 0; i < count; i++) {
         msg += obj[i].value + " - ";
         msg += obj[i].text + "\n";
      }
   }
   else {
      var count = select2.length;
      for (var i = 0; i < count; i++) {
         if (select2.options[i].selected) {
            msg += select2.options[i].value + " - ";
            msg += select2.options[i].text + "\n";
         }
      }
   }
   window.alert(msg);
}
</script>
<b>Список со множественным выбором:</b><br><br>
<select name="select2" id="select2" size="5" multiple>
```

```
<option value="1" selected>Элемент1</option>
<option value="2">Элемент2</option>
<option value="3">Элемент3</option>
<option value="4">Элемент4</option>
<option value="5">Элемент5</option>
<option value="6">Элемент6</option>
</select><br>
<input type="button" value="Значения списка"
       onclick="multi()"><br><br>


<!-- Взаимосвязанные списки  -->

<script>
var arr = [];
arr[1] = [ "Тема1 Элемент1", "Тема1 Элемент2" ];
arr[2] = [ "Тема2 Элемент1", "Тема2 Элемент2", "Тема2 Элемент3" ];
var value1 = [];
value1[1] = [ "1", "2" ];
value1[2] = [ "3", "4", "5" ];
function change1() {
   var index = document.getElementById("select3").value;
   var select4 = document.getElementById("select4");
   var count = arr[index].length;
   select4.length = count;
   for (i = 0; i < count; i++) {
      select4.options[i].value = value1[index][i];
      select4.options[i].text = arr[index][i];
   }
}
function change2() {
   var sel = document.getElementById("select4");
   var msg = "Значение: " + sel.options[sel.selectedIndex].value;
   msg += "\nТекст: " + sel.options[sel.selectedIndex].text;
   window.alert(msg);
}
</script>
<b>Взаимосвязанные списки:</b><br><br>
<select name="select3" id="select3" size="5" onchange="change1()">
<option value="1">Тема1</option>
<option value="2">Тема2</option>
</select><br>
<select name="select4" id="select4" onchange="change2()">
<option value="1" selected>Тема1 Элемент1</option>
<option value="2">Тема1 Элемент2</option>
</select><br><br>
```

```
<!-- Переход на указанный сайт -->

<b>Переход на указанный сайт:</b><br><br>
<select
onchange="top.location.href=this.options[this.selectedIndex].value;">
<option value="http://www.mail.ru/" selected>Национальная почта Mail.ru
</option>
<option value="http://www.rambler.ru/">Рамблер</option>
</select>
</div>
</form>
</body>
</html>
```

3.15.9. Флажок и переключатели

Флажки и переключатели имеют следующие свойства:

- `value` — значение текущего элемента формы;
- `checked` — признак отметки: `true`, если флажок или переключатель находится во включенном состоянии;
- `defaultChecked` — флажок или переключатель установлен по умолчанию. Возвращает `true` или `false`;
- `disabled` — признак запрета: если задано значение `true`, то элемент является неактивным (отображается серым цветом);
- `indeterminate` — если `true`, то флажок находится в неопределенном состоянии, и `false` — в противном случае;
- `form` — ссылка на форму, в которой находится элемент;
- `name` — имя элемента;
- `type` — тип элемента формы.

Методы:

- `blur()` — убирает фокус ввода с текущего элемента формы;
- `focus()` — помещает фокус на текущий элемент формы.

События:

- `onblur` — наступает при потере фокуса элементом формы;
- `onclick` — возникает при выборе элемента;
- `onfocus` — происходит при получении фокуса ввода элементом формы.

Чтобы найти выбранный элемент-переключатель в группе, необходимо перебрать все переключатели в цикле. Получить значение выбранного переключателя можно, указав внутри квадратных скобок индекс элемента в группе. Рассмотрим это на примере (листинг 3.64).

Листинг 3.64. Обработка флажков и переключателей

```
<!DOCTYPE html>
<html lang="ru">
<head>
 <meta charset="utf-8">
 <title>Пример использования флажков и переключателей</title>
<script>
function btnClick() {
   var msg = "";
   if (document.getElementById("check1").checked) {
      msg = "Флажок установлен\n";
      msg += "Значение: " + document.getElementById("check1").value+"\n";
   }
   else {
      msg = "Флажок снят\n";
   }
   var value1 = "";
   var count = document.frm.radio1.length;
   for (i = 0; i < count; i++) {
      if (document.frm.radio1[i].checked) {
         value1 = document.frm.radio1[i].value;
         break;
      }
   }
   if (value1 == "male") {
      msg += "Пол: Мужской\n";
   }
   else {
      msg += "Пол: Женский\n";
   }
   window.alert(msg);
}
</script>
</head>
<body>
<form action="test.php" method="GET" name="frm" id="frm">
<div>
<input type="checkbox" name="check1" id="check1" value="yes" checked>
Текст<br><br>
Укажите ваш пол:<br>
<input type="radio" name="radio1" id="radio1" value="male"
checked>Мужской
<input type="radio" name="radio1" id="radio2" value="female">Женский
<br><br>
<input type="button" value="Вывести значения" onclick="btnClick()">
</div>
```

```
</form>
</body>
</html>
```

3.15.10. Кнопки. Обработка нажатия кнопки

Кнопки поддерживают следующие свойства:

- `value` — текст, отображаемый на кнопке;
- `disabled` — признак запрета: если задано значение `true`, то кнопка является неактивной (отображается серым цветом);
- `form` — ссылка на форму, в которой находится элемент;
- `name` — имя элемента;
- `type` — тип элемента формы.

Методы традиционны:

- `blur()` — убирает фокус ввода с текущего элемента формы;
- `focus()` — помещает фокус на текущий элемент формы.

События:

- `onblur` — наступает при потере фокуса элементом формы;
- `onclick` — возникает при нажатии кнопки;
- `onfocus` — происходит при получении фокуса ввода элементом формы.

В приведенном далее примере (листинг 3.65) кнопка изначально не активна. При вводе в текстовое поле кнопка активируется. По нажатию кнопки текст, введенный в текстовое поле, отображается на кнопке, текстовое поле очищается, и кнопка деактивируется.

Листинг 3.65. Обработка нажатия кнопки

```
<!DOCTYPE html>
<html lang="ru">
<head>
 <meta charset="utf-8">
 <title>Пример использования кнопок</title>
<script>
function keyUp() {
   if (document.getElementById("text1").value == "") {
      document.getElementById("button1").disabled = true;
   }
   else {
      document.getElementById("button1").disabled = false;
   }
}
```

```
function btnClick() {
   document.getElementById("button1").value =
        document.getElementById("text1").value;
   document.getElementById("text1").value = "";
   document.getElementById("button1").disabled = true;
}
</script>
</head>
<body>
 <form action="test.php" method="GET" onsubmit="return false;">
  <div>
  <input type="text" name="text1" id="text1" onkeyup="keyUp()"><br>
  <input type="button" value="Изменить текст на кнопке"
         onclick="btnClick()" id="button1" disabled>
  </div>
 </form>
</body>
</html>
```

Обычная командная кнопка может быть вставлена в Web-страницу не только с помощью тега <input>, но и с помощью парного тега <button>. При использовании этого тега текст на кнопке можно сделать цветным, а также можно задать клавишу быстрого доступа.

Переделаем пример из листинга 3.65 и вместо тега <input> используем тег <button> (листинг 3.66).

Листинг 3.66. Использование тега <button>

```
<!DOCTYPE html>
<html lang="ru">
<head>
 <meta charset="utf-8">
 <title>Пример использования тега &lt;button&gt;</title>
<script>
function keyUp() {
   if (document.getElementById("text1").value == "") {
      document.getElementById("button1").disabled = true;
  }
  else {
     document.getElementById("button1").disabled = false;
  }
}
function btnClick() {
   document.getElementById("span1").textContent =
      document.getElementById("text1").value;
```

```
    document.getElementById("text1").value = "";
    document.getElementById("button1").disabled = true;
}
</script>
</head>
<body>
<form action="test.php" method="GET" onsubmit="return false;">
<div>
<input type="text" name="text1" id="text1" onkeyup="keyUp()"><br>
<button onclick="btnClick()" id="button1" disabled>
<span id="span1" style="color: red">
<span style="text-decoration: underline">Т</span>екст красного цвета
</span></button>
</div>
</form>
</body>
</html>
```

3.15.11. Работа с элементами управления

Элементы управления поддерживают следующие вновь введенные свойства:

- `autocomplete` — аналогично одноименному свойству формы;
- `autofocus` — признак того, должен ли этот элемент управления получить фокус ввода сразу после загрузки страницы. Значение `true` активизирует автоматическое получение фокуса ввода, значение `false` деактивирует его;
- `max` — максимальное значение, которое можно указать в поле ввода числа или выбора значения из диапазона;
- `min` — минимальное значение, которое можно указать в поле ввода числа или выбора значения из диапазона;
- `step` — интервал между значениями, допустимыми для занесения в поле ввода числа или выбора значения из диапазона;
- `pattern` — регулярное выражение, определяющее формат заносимого в поле ввода значения и записанное в виде строки;
- `placeholder` — текст подсказки, выводимой прямо в поле ввода, в виде строки;
- `required` — признак того, является ли элемент управления обязательным для ввода: `true` — является, `false` — не является;
- `selectionStart` — номер первого символа выделенного в поле ввода или в области редактирования фрагмента текста (в виде числа). Если текст не выделен, возвращает номер символа, на котором стоит текстовый курсор;
- `selectionEnd` — номер последнего символа выделенного фрагмента текста (в виде числа). Если текст не выделен, возвращает номер символа, на котором стоит текстовый курсор;

- validationMessage — возвращает текст сообщения (в виде строки) о некорректно занесенном в поле ввода значении;
- validity — возвращает сведения об ошибках, допущенных посетителем при занесении значения в поле ввода. Представляет собой объект класса ValidityState, поддерживающего следующие доступные только для чтения свойства:
  - badInput — true, если введенное посетителем значение неполное;
  - patternMismatch — true, если введенное значение не совпадает с заданным шаблоном (регулярным выражением, занесенным в свойство pattern);
  - rangeOverflow — true, если введенное число больше указанного максимального значения;
  - rangeUnderflow — true, если введенное число меньше указанного минимального значения;
  - stepMismatch — true, если введенное число не укладывается в заданный интервал;
  - tooLong — true, если введенная строка слишком длинная;
  - typeMismatch — true, если введенное значение не соответствует требуемому типу (т. е. не является числом, интернет-адресом, адресом электронной почты и т. п.);
  - valueMissing — true, если обязательное поле пустое;
  - customError — true, если было задано иное сообщение об ошибке ввода данных (как это сделать, будет описано далее);
  - valid — true, если введенное значение полностью корректно (и все приведенные ранее свойства хранят значение false);
- willValidate — возвращает true, если значение, присутствующее в этом элементе управления, будет проверяться на корректность. Для элементов, недоступных для посетителя и доступных лишь для чтения, возвращается false.

3.15.12. Расширенная проверка значения, занесенного в поле ввода

Мы уже знаем, что DOM 3 оснастила поля ввода богатым набором инструментов для проверки занесенных в них данных на корректность. Теперь, если мы хотим удостовериться, что посетитель ввел в поле, скажем, число, а не что-то иное, мы создадим поле ввода числа и укажем для него параметры, которым должно удовлетворять это число, — об остальном позаботится сам Web-браузер.

Если мы хотим вывести свое собственное сообщение об ошибке прямо под элементом управления, в который было введено некорректное значение, как это делает сам Web-браузер, то зададим это сообщение с помощью метода setCustomValidity(<Сообщение>) нужного элемента управления. Как только мы зададим для элемента управления свое сообщение об ошибке, вызвав метод

`setCustomValidity()`, занесенное в этот элемент значение будет считаться некорректным. Web-браузер выведет заданное нами сообщение на экран, а данные отправлены не будут.

Чтобы пометить введенное в элемент управления значение как корректное, достаточно вызвать этот метод, передав ему в качестве параметра пустую строку.

Пример кода, выводящего сообщение об ошибке ввода подобным образом, показан в листинге 3.67.

Листинг 3.67. Пример вывода сообщения об ошибке

```
<input type="text" id="txtYear">
<script>
var txtYear = document.getElementById("txtYear");
txtYear.addEventListener("input", function() {
  var n = parseInt(txtYear.value, 10);
  if (!isNaN(n)) {
    if (n < 2000) {
      txtYear.setCustomValidity(
            "Значение года не должно быть меньше 2000");
    }
    else {
       txtYear.setCustomValidity("");
    }
  }
}, false);
</script>
```

3.16. Работа с графическими изображениями

Для управления графическими изображениями DOM 3 предоставляет нам три дополнительных свойства, поддерживаемые объектом-изображением:

- `complete` — возвращает `true`, если изображение было полностью загружено, и `false` — в противном случае;
- `naturalWidth` — возвращает ширину загруженного изображения (в пикселах). Значение отличается от значения параметра `width`, который задает отображаемую ширину изображения;
- `naturalHeight` — возвращает высоту загруженного изображения (в пикселах). Значение отличается от значения параметра `height`, который задает отображаемую высоту изображения.

Пример:

```
<img src="photo.jpg" width="200" alt="" id="img">
<script>
window.onload = function() {
```

```
    var img = document.getElementById("img");
    console.log(img.complete);
    console.log(img.naturalWidth);
    console.log(img.naturalHeight);
    console.log(img.width);
    console.log(img.height);
}
</script>
```

3.17. Работа с мультимедиа

Элементы аудио- и видеороликов, помещенных на страницу средствами HTML 5, поддерживают расширенный набор программных инструментов, который довольно велик.

3.17.1. Свойства аудио- и видеороликов

Начнем, как обычно, со списка дополнительных свойств:

- `src` — интернет-адрес воспроизводящегося в текущий момент ролика;
- `currentSrc` — возвращает интернет-адрес воспроизводящегося в текущий момент ролика. Может отличаться от интернет-адреса, заданного в самом теге, если ролик предоставляется серверным приложением, автоматически выбирающим мультимедийный файл в зависимости от параметров клиентского компьютера;
- `poster` — интернет-адрес файла заставки для видео;
- `preload` — признак, следует ли выполнять предварительную загрузку ролика. Значением этого свойства должна быть одна из строк:
  - `"none"` — не выполнять предзагрузку;
  - `"metadata"` — загрузить только самое начало файла, где хранятся сведения о ролике;
  - `"auto"` — загрузить весь файл (значение по умолчанию);
- `readyState` — возвращает состояние загрузки ролика (в виде числа):
  - `0` — загрузка еще не началась;
  - `1` — загружен лишь заголовок файла, хранящий сведения о самом ролике;
  - `2` — загружены данные, достаточные для воспроизведения текущего кадра, но для воспроизведения следующих кадров данных недостаточно;
  - `3` — загружены данные, достаточные для воспроизведения текущего и следующих кадров;
  - `4` — загружены данные, достаточные для бесперебойного воспроизведения всего ролика;

- `networkState` — возвращает текущее состояние загрузки ролика (в виде числа):
  - `0` — загрузка еще не началась;
  - `1` — ролик уже загружен;
  - `2` — идет загрузка ролика;
  - `3` — файл с роликом отсутствует на сервере;
- `autoplay` — признак, запустится ли воспроизведение ролика сразу после его загрузки: `true` — запустится, `false` — не запустится;
- `width` — ширина видеоролика (в пикселах);
- `height` — высота видеоролика (в пикселах);
- `videoWidth` — возвращает ширину загруженного видеоролика (в пикселах);
- `videoHeight` — возвращает высоту загруженного видеоролика (в пикселах);
- `controls` — признак, будут ли на экране присутствовать элементы управления воспроизведением ролика: `true` — будут присутствовать, `false` — не будут присутствовать;
- `ended` — возвращает `true`, если воспроизведение ролика закончилось, и `false` — в противном случае;
- `duration` — возвращает продолжительность ролика в виде числа в секундах;
- `currentTime` — текущая позиция воспроизведения ролика в виде числа в секундах;
- `loop` — признак, будет ли ролик воспроизводиться бесконечно: `true` — ролик воспроизводится бесконечно, `false` — ролик будет воспроизведен всего один раз;
- `paused` — признак, приостановлено ли воспроизведение ролика в текущий момент: `true` — приостановлено, `false` — не приостановлено;
- `playbackRate` — текущая скорость воспроизведения ролика. Значение должно представлять собой число с плавающей точкой, которое будет умножено на значение скорости воспроизведения, полученной из файла с роликом: положительные значения задают воспроизведение в прямом порядке, отрицательные — в обратном;
- `volume` — громкость звука. Значение должно представлять собой число с плавающей точкой от `0` (звук отсутствует) до `1` (максимальная громкость);
- `muted` — признак, приглушен ли звук в настоящий момент: `true` — приглушен, `false` — не приглушен;
- `seeking` — возвращает `true`, если в настоящий момент посетитель меняет позицию воспроизведения ролика, и `false` — в противном случае.

3.17.2. Методы аудио- и видеороликов

Метод `canPlayType(<MIME-тип>)` позволяет узнать, поддерживает ли Web-браузер воспроизведение роликов указанного формата. MIME-тип формата ролика указывается в виде строки. Метод возвращает одно из следующих строковых значений:

- `"probably"` — скорее всего, поддерживает;
- `"maybe"` — возможно, поддерживает;
- `""` (пустая строка) — гарантированно не поддерживает.

Вот еще три метода, не принимающие параметров и не возвращающие результат:

- `load()` — выполняет повторную загрузку ролика;
- `pause()` — приостанавливает воспроизведение ролика;
- `play()` — запускает или возобновляет воспроизведение ролика.

3.17.3. События аудио- и видеороликов

Приведем события аудио- и видеороликов:

- `onabort` — возникает, когда посетитель прерывает загрузку файла с роликом;
- `oncanplay` — возникает, если Web-браузер получил достаточно данных, чтобы, по крайней мере, начать воспроизведение ролика, однако в будущем возможны приостановки воспроизведения для подгрузки данных;
- `oncanplaythrough` — возникает, если Web-браузер получил достаточно данных, чтобы начать воспроизведение ролика без приостановок для подгрузки данных;
- `ondurationchange` — возникает, когда Web-браузер получает значение продолжительности ролика или когда это значение почему-то изменяется;
- `onemptied` — возникает при выгрузке файла ролика, что может произойти, например, при вызове метода `load()`;
- `onended` — возникает, если воспроизведение ролика закончилось;
- `onerror` — возникает при прерывании загрузки файла с роликом в результате ошибки скрипта или сетевого сбоя;
- `onloadeddata` — возникает, если Web-браузер загрузил достаточно данных для вывода на экран первого кадра, но не для начала воспроизведения ролика;
- `onloadedmetadata` — возникает после загрузки метаданных;
- `onloadstart` — возникает, когда Web-браузер только начинает загрузку файла с роликом;
- `onpause` — возникает, когда воспроизведение ролика приостанавливается;
- `onplay` — возникает после начала или возобновления воспроизведения ролика;
- `onplaying` — возникает, когда Web-браузер получает достаточно данных, чтобы возобновить воспроизведение приостановленного ролика;

- `onprogress` — периодически возникает в процессе загрузки Web-браузером мультимедийного файла;
- `onratechange` — возникает при изменении значения скорости воспроизведения ролика;
- `onseeked` — возникает по завершении изменения текущей позиции воспроизведения ролика;
- `onseeking` — возникает после начала изменения текущей позиции воспроизведения ролика;
- `onstalled` — возникает через три секунды после остановки процесса подгрузки очередной порции данных из мультимедийного файла. Обычно означает, что дальнейшая подгрузка файла с роликом невозможна;
- `ontimeupdate` — возникает при изменении текущей позиции воспроизведения;
- `onvolumechange` — возникает при изменении значения громкости, приглушении звука или восстановления его после приглушения;
- `onwaiting` — возникает, когда воспроизведение ролика приостанавливается для подгрузки очередной порции данных.

3.18. Холст в HTML 5. Программируемая графика

В *главе 1* упоминалось, что язык HTML 5 предлагает довольно богатые возможности по выводу на страницу произвольной графики, рисуемой программно, в скрипте. Настала пора узнать, как это делается.

3.18.1. Тег <*canvas*>

Холст создается с помощью парного тега `<canvas>`, имеющего следующие параметры:

- `width` — ширина холста;
- `height` — высота холста.

Если параметры не указаны, то создается холст с размерами 300×150 пикселов.

```
<canvas id="cnv" width="400" height="300">
<p>Ваш Web-браузер не поддерживает элемент canvas</p>
</canvas>
```

3.18.2. Создание контекста рисования

Создав на странице сам холст, мы можем получить его *контекст рисования* — набор инструментов, применяемых для рисования на нем. Для этого используется метод `getContext("2d")`:

```
var oCanvas = document.getElementById("cnv");
var ctxCanvas = oCanvas.getContext("2d");
```

Контекст рисования представляется объектом класса `CanvasRenderingContext2D`. Все операции по рисованию выполняются с применением его свойств и методов, которые мы далее рассмотрим.

Получить ссылку на объект холста позволяет свойство `canvas`:

```
console.log(ctxCanvas.canvas.height); // 300
```

3.18.3. Изменение характеристик заливки

Свойство `fillStyle` определяет цвет заливки, который может быть задан любым поддерживаемым CSS способом. По умолчанию цвет заливки черный.

- Пример указания синего цвета заливки и рисования прямоугольника:

  ```
  ctxCanvas.fillStyle = "rgb(0, 0, 255)";
  ctxCanvas.fillRect(50, 50, 200, 200);
  ```

- Пример указания полупрозрачного красного цвета и рисования прямоугольника:

  ```
  ctxCanvas.fillStyle = "rgba(255, 0, 0, 0.5)";
  ctxCanvas.fillRect(100, 100, 150, 150);
  ```

3.18.4. Изменение характеристик обводки

Цвет обводки задается с помощью свойства `strokeStyle`. Сам цвет может быть указан любым способом, поддерживаемым CSS. По умолчанию цвет обводки черный.

- Пример указания синего цвета обводки и рисования прямоугольника:

  ```
  ctxCanvas.strokeStyle = "rgb(0, 0, 255)";
  ctxCanvas.strokeRect(50, 50, 200, 200);
  ```

- Пример указания полупрозрачного красного цвета обводки и рисования прямоугольника:

  ```
  ctxCanvas.strokeStyle = "rgba(255, 0, 0, 0.5)";
  ctxCanvas.strokeRect(100, 100, 150, 150);
  ```

Управлять характеристиками обводки позволяют следующие свойства и методы:

- `lineWidth` — задает толщину линий обводки (в пикселах):

  ```
  ctxCanvas.lineWidth = 20;
  ctxCanvas.strokeRect(50, 50, 200, 200);
  ```

- `lineCap` — задает форму окончания линии:
  - `"square"` — квадратные концы (прибавляются к длине линии);
  - `"butt"` — концы никак не оформляются; значение по умолчанию;
  - `"round"` — закругленные концы (прибавляются к длине линии).

Пример рисования прямой линии с закругленными концами:

```
ctxCanvas.beginPath();
ctxCanvas.lineWidth = 10;
ctxCanvas.lineCap = "round";
ctxCanvas.moveTo(20, 20);
ctxCanvas.lineTo(180, 20);
ctxCanvas.stroke();
```

- `lineJoin` — задает форму окончания в месте соединения двух линий обводки:
  - `"miter"` — обычные углы; значение по умолчанию;
  - `"bevel"` — скошенные углы;
  - `"round"` — закругленные углы.

  Пример рисования треугольника со скругленными углами:

```
ctxCanvas.beginPath();
ctxCanvas.lineWidth = 15;
ctxCanvas.lineJoin = "round";
ctxCanvas.moveTo(200, 20);
ctxCanvas.lineTo(20, 100);
ctxCanvas.lineTo(200, 100);
ctxCanvas.closePath();
ctxCanvas.stroke();
```

- `miterLimit` — задает ограничение длины угла, когда для свойства `lineJoin` задано значение `"miter"`. Значение по умолчанию: `10` пикселов. Если значение превышено, то угол будет скошенным;
- `lineDashOffset` — задает смещение начала пунктирной обводки. Значение по умолчанию: `0`;
- `setLineDash(<Значения>)` — задает значения для пунктирной линии. Значения указываются в виде массива. Четные индексы задают длину штриха, а нечетные — длину пропуска.

  Пример рисования пунктирной линии:

```
ctxCanvas.beginPath();
ctxCanvas.lineWidth = 5;
ctxCanvas.lineDashOffset = 10;
ctxCanvas.setLineDash([15.0, 10.0]);
ctxCanvas.miterLimit = 20;
ctxCanvas.moveTo(200, 20);
ctxCanvas.lineTo(50, 200);
ctxCanvas.stroke();
```

  Чтобы сделать следующую линию опять сплошной, достаточно передать в метод `setLineDash()` пустой массив;
- `getLineDash()` — возвращает массив со значениями для пунктирной линии.

3.18.5. Рисование прямоугольников

Для рисования прямоугольников предназначены следующие методы:

- `fillRect()` — рисует прямоугольник, используя характеристики заливки. Формат метода:

  ```
  fillRect(<X>, <Y>, <Ширина>, <Высота>)
  ```

  Параметры `<X>` и `<Y>` задают координаты левого верхнего угла прямоугольника. Положительная ось `X` направлена вправо, а положительная ось `Y` — вниз:

  ```
  ctxCanvas.fillStyle = "rgb(0, 0, 255)";
  ctxCanvas.fillRect(50, 50, 200, 200);
  ```

- `strokeRect()` — рисует прямоугольник, используя характеристики обводки. Формат метода:

  ```
  strokeRect(<X>, <Y>, <Ширина>, <Высота>)
  ```

  Пример:

  ```
  ctxCanvas.strokeStyle = "rgb(0, 0, 255)";
  ctxCanvas.strokeRect(50, 50, 200, 200);
  ```

3.18.6. Очистка прямоугольной области или всего холста

Метод `clearRect()` очищает заданную прямоугольную область от любой присутствовавшей там графики. Формат метода:

```
clearRect(<X>, <Y>, <Ширина>, <Высота>)
```

- Пример рисования прямоугольника с заливкой и очистки прямоугольной области внутри него:

  ```
  ctxCanvas.fillRect(0, 0, 400, 300);
  ctxCanvas.clearRect(100, 100, 200, 100);
  ```

- Пример рисования прямоугольника с заливкой, очистки всего холста и опять рисования прямоугольника с заливкой:

  ```
  ctxCanvas.fillRect(0, 0, 400, 300);
  ctxCanvas.clearRect(0, 0, ctxCanvas.canvas.width,
                            ctxCanvas.canvas.height);
  ctxCanvas.fillRect(50, 50, 100, 100);
  ```

3.18.7. Вывод текста

Свойство `font` задает параметры шрифта, которым будет выводиться текст. Эти параметры указывают в том же формате, что и у значения атрибута CSS `font`, в виде строки:

```
ctxCanvas.font = "italic 16pt Verdana";
ctxCanvas.strokeText("JavaScript", 200, 50, 100);
```

Для вывода текста используются следующие методы:

- `strokeText()` — выводит текст, используя характеристики обводки. Формат метода:

  ```
  strokeText(<Текст>, <X>, <Y>[, <Максимальная ширина>])
  ```

  С первыми тремя параметрами все ясно. Четвертый, необязательный, параметр определяет максимальное значение ширины, которую может принять выводимый текст. Если выводимый текст получается шире, Web-браузер выводит его либо шрифтом с уменьшенной шириной символов (если выбранный шрифт поддерживает такое начертание), либо шрифтом меньшего размера:

  ```
  ctxCanvas.strokeStyle = "blue";
  ctxCanvas.font = "italic 16pt Verdana";
  ctxCanvas.strokeText("JavaScript", 200, 50, 50);
  ```

- `fillText()` — выводит текст, используя характеристики заливки. Формат метода:

  ```
  fillText(<Текст>, <X>, <Y>[, <Максимальная ширина>])
  ```

  Пример:

  ```
  ctxCanvas.fillStyle = "blue";
  ctxCanvas.font = "italic 16pt Verdana";
  ctxCanvas.fillText("JavaScript", 200, 50);
  ```

Свойство `textAlign` устанавливает горизонтальное выравнивание выводимого текста относительно точки, в которой он будет выведен. Это свойство может принимать следующие значения:

- `"left"` — выравнивание по левому краю;
- `"right"` — выравнивание по правому краю;
- `"start"` — выравнивание по левому краю, если текст выводится по направлению слева направо, и по правому краю в противном случае (значение по умолчанию);
- `"end"` — выравнивание по правому краю, если текст выводится по направлению слева направо, и по левому краю в противном случае;
- `"center"` — выравнивание по центру:

  ```
  ctxCanvas.textAlign = "center";
  ctxCanvas.fillText("HTML, CSS", 100, 100);
  ```

Свойство `textBaseline` позволяет задать вертикальное выравнивание выводимого текста относительно точки, в которой он будет выведен. Доступны следующие значения:

- `"top"` — выравнивание по верху прописных букв;
- `"hanging"` — выравнивание по верху строчных букв;
- `"middle"` — выравнивание по средней линии строчных букв;
- `"alphabetic"` — выравнивание по базовой линии букв европейских алфавитов (значение по умолчанию);

- "ideographic" — выравнивание по базовой линии иероглифических символов (она находится чуть ниже базовой линии букв европейских алфавитов);
- "bottom" — выравнивание по низу букв:

```
ctxCanvas.textBaseline = "bottom";
ctxCanvas.fillText("HTML, CSS", 100, 100);
```

Еще нам может пригодиться метод measureText(<Текст>), позволяющий узнать ширину текста. Метод возвращает объект с единственным свойством width, которое и хранит ширину текста в пикселах:

```
var s = "HTML, CSS, JavaScript, PHP, MySQL";
ctxCanvas.font = "bold 24pt Tahoma";
console.log(ctxCanvas.measureText(s).width);
```

3.18.8. Вывод изображения

Вывести изображение на холст позволяет метод drawImage(). Форматы метода:

```
drawImage(<img>, <x>, <y>)
drawImage(<img>, <x>, <y>, <w>, <h>)
drawImage(<img>, <sx>, <sy>, <sw>, <sh>,
                 <dx>, <dy>, <dw>, <dh>)
```

Первый формат выводит изображение полностью в позицию с координатами <x> и <y>. В параметре <img> можно указать изображение, видео или объект другого холста:

```
<canvas id="cnv" width="400" height="300"></canvas>
<canvas id="cnv2" width="400" height="300"></canvas>
<img alt="" src="photo.jpg" id="img">
<script>
var oCanvas = document.getElementById("cnv");
var ctxCanvas = oCanvas.getContext("2d");

var imgSample = new Image();
imgSample.src = "photo.jpg";
ctxCanvas.drawImage(imgSample, 0, 0);

var oCanvas2 = document.getElementById("cnv2");
var ctxCanvas2 = oCanvas2.getContext("2d");
ctxCanvas2.fillStyle = "rgb(0, 0, 255)";
ctxCanvas2.fillRect(50, 50, 200, 200);
ctxCanvas.drawImage(oCanvas2, 0, 0);

var oImage = document.getElementById("img");
ctxCanvas2.drawImage(oImage, 0, 0);
</script>
```

Второй формат позволяет ограничить область вывода шириной (параметр <w>) и высотой (параметр <h>). При этом изображение может быть уменьшено или уве-

личено таким образом, чтобы вписаться в область. Если пропорции области не совпадают с пропорциями изображения, то изображение будет растянуто или сжато без соблюдения пропорций:

```
var img = new Image();
img.src = "photo.jpg";
ctxCanvas.drawImage(img, 0, 0, 400, 100);
```

Третий формат берет прямоугольную область (<sx>, <sy>, <sw>, <sh>) из изображения и вписывает ее в прямоугольную область (<dx>, <dy>, <dw>, <dh>) на холсте:

```
var img = new Image();
img.src = "photo.jpg";
ctxCanvas.drawImage(img, 0.0,   0.0,   50.0,  50.0,
                         200.0, 20.0,  50.0,  50.0);
ctxCanvas.drawImage(img, 0.0,   0.0,   50.0,  50.0,
                         200.0, 100.0, 100.0, 100.0);
```

3.18.9. Рисование траектории

Управлять рисованием траектории позволяют следующие методы:

- `beginPath()` — начинает новую траекторию;
- `moveTo(<x>, <y>)` — позволяет переместить текущую позицию в точку с указанными координатами;
- `closePath()` — позволяет замкнуть текущую траекторию;
- `stroke()` — прорисовывает текущую траекторию, используя характеристики обводки;
- `fill()` — прорисовывает текущую траекторию, используя характеристики заливки.

  Пример рисования замкнутого треугольника с обводкой и заливкой:

  ```
  ctxCanvas.fillStyle = "rgb(0, 0, 255)";
  ctxCanvas.strokeStyle = "rgb(255, 0, 0)";
  ctxCanvas.lineWidth = 3;
  ctxCanvas.beginPath();
  ctxCanvas.moveTo(20, 20);
  ctxCanvas.lineTo(100, 20);
  ctxCanvas.lineTo(100, 80);
  ctxCanvas.closePath();
  ctxCanvas.stroke();
  ctxCanvas.fill();
  ```

- `lineTo(<x>, <y>)` — добавляет прямую линию к текущей траектории:

  ```
  ctxCanvas.lineTo(100, 20);
  ```

- `arcTo()` — добавляет дугу к текущей траектории. Формат метода:

  ```
  arcTo(<x1>, <y1>, <x2>, <y2>, <radius>)
  ```

Пример рисования дуги:

```
ctxCanvas.strokeStyle = "rgb(0, 0, 0)";
ctxCanvas.lineWidth = 5;
ctxCanvas.beginPath();
ctxCanvas.moveTo(100, 100);
ctxCanvas.arcTo(200, 0, 300, 100, 100);
ctxCanvas.stroke();
```

- `arc()` — добавляет дугу к текущей траектории. Формат метода:

```
arc(<centerX>, <centerY>, <radius>, <startAngle>, <endAngle>,
    <true | false>)
```

Параметры `<centerX>` и `<centerY>` задают координаты центра круга, параметр `<radius>` — радиус, параметр `<startAngle>` — начальный угол в радианах, а параметр `<endAngle>` — конечный угол в радианах. Если шестой параметр имеет значение `true`, то дуга рисуется против часовой стрелки, а если `false` — по часовой стрелке.

Тот факт, что углы задаются в радианах, несколько осложняет работу. Нам придется пересчитывать величины углов из градусов в радианы с помощью следующего выражения:

```
radians = (Math.PI / 180) * degrees;
```

Здесь переменная `degrees` хранит значение угла в градусах, а переменная `radians` будет хранить то же значение, но в радианах.

Пример рисования круга:

```
ctxCanvas.strokeStyle = "rgb(0, 0, 0)";
ctxCanvas.lineWidth = 5;
ctxCanvas.beginPath();
ctxCanvas.arc(200, 150, 100, 0, Math.PI * 2, false);
ctxCanvas.stroke();
```

- `bezierCurveTo()` — добавляет кубическую кривую Безье к текущей траектории. Формат метода:

```
bezierCurveTo(<xc1>, <yc1>, <xc2>, <yc2>, <x1>, <y1>)
```

Параметры `<xc1>` и `<yc1>` задают координаты первой опорной точки, параметры `<xc2>` и `<yc2>` — координаты второй опорной точки, а параметры `<x1>` и `<y1>` — координаты конечной точки:

```
ctxCanvas.beginPath();
ctxCanvas.moveTo(100, 100);
ctxCanvas.bezierCurveTo(120, 80, 160, 20, 100, 200);
ctxCanvas.stroke();
```

- `quadraticCurveTo()` — добавляет квадратичную кривую к текущей траектории. Формат метода:

```
quadraticCurveTo(<xc>, <yc>, <x1>, <y1>)
```

Параметры `<xc>` и `<yc>` задают координаты опорной точки, а параметры `<x1>` и `<y1>` — координаты конечной точки:

```
ctxCanvas.beginPath();
ctxCanvas.moveTo(100, 100);
ctxCanvas.quadraticCurveTo(200, 100, 200, 200);
ctxCanvas.stroke();
```

- `rect()` — добавляет к траектории прямоугольник. Формат метода:

```
rect(<x>, <y>, <w>, <h>)
```

Нарисуем фигуру, состоящую их трех накладывающихся друг на друга разноцветных квадратов:

```
ctxCanvas.beginPath();
ctxCanvas.fillStyle = "red";
ctxCanvas.rect(50, 50, 50, 50);
ctxCanvas.fill();
ctxCanvas.beginPath();
ctxCanvas.fillStyle = "green";
ctxCanvas.rect(75, 75, 50, 50);
ctxCanvas.fill();
ctxCanvas.beginPath();
ctxCanvas.fillStyle = "blue";
ctxCanvas.rect(100, 100, 50, 50);
ctxCanvas.fill();
```

- `clip()` — создает маску на основе текущего пути (путь должен быть закрытым). Область внутри пути будет видна, а вне — скрыта:

```
ctxCanvas.beginPath();
ctxCanvas.moveTo(100, 150);
ctxCanvas.lineTo(200, 0);
ctxCanvas.lineTo(200, 300);
ctxCanvas.closePath();
ctxCanvas.clip();
ctxCanvas.fillRect(0, 100, 400, 100);
```

3.18.10. Определение вхождения точки в состав контура

Иногда бывает необходимо выяснить, входит ли точка с заданными координатами в состав контура сложной фигуры. Сделать это мы можем с помощью метода `isPointInPath(<x>, <y>)`. Метод возвращает `true`, если точка с такими координатами входит в состав контура, и `false` — в противном случае:

```
ctxCanvas.beginPath();
ctxCanvas.rect(50, 50, 50, 50);
ctxCanvas.stroke();
```

```
if (ctxCanvas.isPointInPath(60, 50)) {
   window.alert("Точка входит в состав контура");
}
```

3.18.11. Использование сложных цветов

Помимо однотонных цветов, холст позволяет использовать для закрашивания линий и заливок градиенты и даже выполнять заливку текстурой.

Линейный градиент

Линейный градиент создают в три этапа. Первый этап — вызов метода `createLinearGradient()` — собственно создание линейного градиента:

```
createLinearGradient(<x1>, <y1>, <x2>, <y2>)
```

Метод `createLinearGradient()` возвращает объект класса `CanvasGradient`, представляющий созданный линейный градиент. С его помощью мы указываем цвета, формирующие градиент.

Второй этап — расстановка ключевых точек градиента. Здесь нам понадобится метод `addColorStop()` класса `CanvasGradient`:

```
<Градиент>.addColorStop(<Положение ключевой точки>, <Цвет>)
```

Первый параметр задается в виде числа от `0.0` (начало прямой) до `1.0` (конец прямой). Результат этот метод не возвращает.

Третий этап — использование готового линейного градиента. Для этого представляющий его объект класса `CanvasGradient` следует присвоить свойству `strokeStyle` или `fillStyle`.

Пример:

```
var lgSample = ctxCanvas.createLinearGradient(0, 0, 100, 50);
lgSample.addColorStop(0, "black");
lgSample.addColorStop(0.4, "rgba(0, 0, 255, 0.5)");
lgSample.addColorStop(1, "#ff0000");
ctxCanvas.fillStyle = lgSample;
ctxCanvas.fillRect(0, 0, 200, 100);
```

Радиальный градиент

Радиальный градиент также создают в три этапа. Первый этап — вызов метода `createRadialGradient()` — создание радиального градиента:

```
createRadialGradient(
    <Горизонтальная координата центра внутренней окружности>,
    <Вертикальная координата центра внутренней окружности>,
    <Радиус внутренней окружности>,
    <Горизонтальная координата центра внешней окружности>,
    <Вертикальная координата центра внешней окружности>,
    <Радиус внешней окружности>)
```

Параметры этого метода определяют координаты центров и радиусы обеих окружностей, описывающих радиальный градиент. Они задаются в пикселах в виде чисел.

Метод `createRadialGradient()` возвращает объект класса `CanvasGradient`, представляющий созданный нами радиальный градиент.

Второй этап — расстановка ключевых точек — выполняется с помощью уже знакомого нам метода `addColorStop()` класса `CanvasGradient`. Только в данном случае первый параметр определит относительное положение создаваемой ключевой точки на промежутке между внутренней и внешней окружностями. Он задается в виде числа от `0.0` (начало промежутка, т. е. внутренняя окружность) до `1.0` (конец промежутка, т. е. внешняя окружность).

И третий этап — использование созданного градиента.

Пример:

```
var rgSample = ctxCanvas.createRadialGradient(100, 100, 10, 150, 100, 120);
rgSample.addColorStop(0, "#cccccc");
rgSample.addColorStop(0.8, "black");
rgSample.addColorStop(1, "#00ff00");
ctxCanvas.fillStyle = rgSample;
ctxCanvas.fillRect(0, 0, 200, 100);
```

Заливка текстурой

Заливка текстурой выполняется с помощью метода `createPattern()`:

```
createPattern(<Графическое изображение или холст>, <Режим повторения>)
```

Первый параметр задает графическое изображение в виде объекта класса `Image`, элемента `IMG` или объекта другого холста.

Часто бывает так, что размеры заданного графического изображения меньше, чем фигуры, к которой должен быть применен графический цвет. В этом случае изображение повторяется столько раз, чтобы полностью «вымостить» линию или заливку. Режим такого повторения задает второй параметр метода `createPattern()`. Его значение должно быть одной из следующих строк:

- `"repeat"` — повтор по горизонтали и вертикали;
- `"repeat-x"` — повтор только по горизонтали;
- `"repeat-y"` — повтор только по вертикали;
- `"no-repeat"` — без повтора.

Метод `createPattern()` возвращает объект класса `CanvasPattern`.

Пример:

```
var imgSample = new Image();
imgSample.src = "graphic_color.jpg";
var cpSample = ctxCanvas.createPattern(imgSample, "repeat");
```

```
ctxCanvas.fillStyle = cpSample;
ctxCanvas.fillRect(0, 0, 200, 100);
```

Текстура не фиксируется на холсте, а полностью применяется к рисуемой фигуре. В этом ее принципиальное отличие от градиентов.

3.18.12. Сохранение и восстановление состояния

Значения основных характеристик можно сохранить в стек, выполнить какую-либо операцию рисования, а затем восстановить эти значения из стека. Для этого предназначены следующие методы:

- `save()` — сохраняет значения основных характеристик (в частности характеристики заливки, обводки, текста и др.) в стек;
- `restore()` — восстанавливает значения основных характеристик из стека.

При сохранении состояния сохраняются:

- все заданные трансформации;
- значения свойств `globalAlpha`, `globalCompositeOperation`, `fillStyle`, `lineCap`, `lineJoin`, `lineWidth`, `miterLimit` и `strokeStyle`;
- все заданные маски.

3.18.13. Трансформации

Выполнить различные преобразования позволяют следующие методы:

- `scale(<dx>, <dy>)` — изменяет масштабирование. Параметры этого метода задают масштаб для горизонтальной и вертикальной осей системы координат в виде чисел. Числа меньше `1.0` задают уменьшение масштаба, а числа больше `1.0` — увеличение. Если нужно оставить масштаб по какой-то из осей неизменным, достаточно указать значение `1.0` соответствующего параметра.

  Вот пример увеличения масштаба в два раза:

  ```
  ctxCanvas.save();
  ctxCanvas.scale(2.0, 2.0);
  ctxCanvas.fillRect(10, 40, 40, 40);
  ctxCanvas.restore();
  ```

- `rotate(<Угол в радианах>)` — применяет трансформацию вращения. По умолчанию вращение выполняется относительно левого верхнего угла холста. Отметим, что поворот всегда выполняется по часовой стрелке:

  ```
  ctxCanvas.save();
  ctxCanvas.translate(200, 150);
  for (var i = 0; i < 3; i++) {
    ctxCanvas.rotate(Math.PI / 6);
    ctxCanvas.strokeRect(-50, -50, 100, 100);
  }
  ctxCanvas.restore();
  ```

- `translate(<tx>, <ty>)` — сдвигает систему координат.

3.18.14. Управление наложением графики

Когда мы рисуем какую-либо фигуру поверх уже существующей, новая фигура накладывается на старую, перекрывая ее. Это поведение по умолчанию, которое мы можем изменить, указав другие значения для свойства `globalCompositeOperation`.

Вот перечень поддерживаемых значений:

- `"source-over"` — новая фигура накладывается на старую, перекрывая ее (значение по умолчанию);
- `"source-atop"` — отображается только та часть новой фигуры, которая накладывается на старую, остальная часть новой фигуры не выводится. Старая фигура выводится целиком и находится ниже новой;
- `"source-in"` — отображается только та часть новой фигуры, которая накладывается на старую, остальные части новой и старой фигур не выводятся;
- `"source-out"` — отображается только та часть новой фигуры, которая не накладывается на старую, остальные части новой фигуры и вся старая фигура не выводятся;
- `"destination-over"` — новая фигура перекрывается старой;
- `"destination-atop"` — отображается только та часть старой фигуры, которая накладывается на новую, остальная часть старой фигуры не выводится. Новая фигура выводится целиком и находится ниже старой;
- `"destination-in"` — отображается только та часть старой фигуры, на которую накладывается новая, остальные части новой и старой фигур не выводятся;
- `"destination-out"` — отображается только та часть старой фигуры, на которую не накладывается новая, остальные части новой фигуры и вся старая фигура не выводятся;
- `"lighter"` — цвета накладывающихся частей старой и новой фигур складываются, результирующий цвет получается более светлым, окрашиваются накладывающиеся части фигур;
- `"xor"` — отображаются только те части старой и новой фигур, которые не накладываются друг на друга;
- `"copy"` — выводится только новая фигура, все старые фигуры удаляются с холста.

Заданный нами способ наложения действует только для графики, которую мы нарисуем после этого. На уже нарисованную графику он не влияет.

Приведенный в следующем примере код рисует два накладывающихся прямоугольника разных цветов и позволит изучить их поведение при разных значениях свойства `globalCompositeOperation` — изменяем значение этого свойства, перезагружаем страницу и смотрим, что получается:

```
ctxCanvas.fillStyle = "blue";
ctxCanvas.fillRect(0, 50, 400, 200);
```

```
ctxCanvas.fillStyle = "red";
ctxCanvas.globalCompositeOperation = "source-over";
ctxCanvas.fillRect(100, 0, 200, 300);
```

3.18.15. Задание уровня прозрачности

Свойство `globalAlpha` указывает уровень прозрачности для любой графики, которую мы впоследствии нарисуем. Уровень прозрачности также задается в виде числа от `0.0` (полностью прозрачный) до `1.0` (полностью непрозрачный; значение по умолчанию):

```
ctxCanvas.globalAlpha = 0.1;
```

3.18.16. Создание тени

Холст позволяет создавать тень у всех рисуемых фигур. Для задания ее параметров применяют четыре свойства:

- `shadowOffsetX` — смещение тени по горизонтали относительно фигуры (в виде числа в пикселах, значение по умолчанию — `0`);
- `shadowOffsetY` — смещение тени по вертикали относительно фигуры (в виде числа в пикселах, значение по умолчанию — `0`);
- `shadowBlur` — степень размытия тени в виде числа: чем больше это число, тем сильнее размыта тень (значение по умолчанию — `0`, т. е. отсутствие размытия);
- `shadowColor` — цвет тени (по умолчанию — черный непрозрачный).

Пример:

```
ctxCanvas.shadowOffsetX = 2;
ctxCanvas.shadowOffsetY = -2;
ctxCanvas.shadowBlur = 4;
ctxCanvas.shadowColor = "rgba(128, 128, 128, 0.5)";
ctxCanvas.fillText("Двое: я и моя тень.", 150, 50);
```

3.18.17. Работа с отдельными пикселами

И наконец, мы можем работать с отдельными пикселами графики, нарисованной на холсте. Это может пригодиться при создании очень сложного изображения или при наложении на графику специальных эффектов наподобие размытия.

Получение массива пикселов

Получить массив пикселов, представляющий нарисованную на холсте графику или ее фрагмент, позволяет метод `getImageData()`. Формат метода:

```
getImageData(<x>, <y>, <Ширина>, <Высота>)
```

Первые два параметра указывают координату левого верхнего угла фрагмента нарисованной графики, два последних — его ширину и высоту.

Метод `getImageData()` возвращает объект класса `ImageData`, представляющий массив пикселов, что составляют присутствующий фрагмент графики с указанными нами параметрами:

```
var idSample = ctxCanvas.getImageData(200, 150, 100, 100);
```

Создание пустого массива пикселов

Чтобы самостоятельно нарисовать какое-либо изображение, можно сначала создать пустой массив пикселов, вызвав метод `createImageData()`:

```
createImageData(<Ширина>, <Высота>)
```

Метод возвращает объект класса `ImageData`:

```
var idEmpty = ctxCanvas.createImageData(400, 300);
```

Здесь мы создаем пустой массив пикселов тех же размеров, что и сам холст.

Манипуляция пикселами

Теперь мы можем начать работать с отдельными пикселами полученного массива, задавая для них цвет и значения полупрозрачности и тем самым формируя какое-либо изображение или изменяя уже существующее.

Класс `ImageData` поддерживает свойство `data`. Его значением является объект-коллекция, хранящая набор чисел:

- первое число представляет собой долю красного цвета в цвете первого пиксела массива;
- второе число — долю зеленого цвета в цвете первого пиксела;
- третье число — долю синего цвета в цвете первого пиксела;
- четвертое число — степень полупрозрачности цвета первого пиксела;
- пятое число — долю красного цвета в цвете второго пиксела;
- шестое число — долю зеленого цвета в цвете второго пиксела;
- седьмое число — долю синего цвета в цвете второго пиксела;
- восьмое число — степень полупрозрачности цвета второго пиксела;
- ...
- n–3-е число — долю красного цвета в цвете последнего пиксела;
- n–2-е число — долю зеленого цвета в цвете последнего пиксела;
- n–1-е число — долю синего цвета в цвете последнего пиксела;
- n-е число — степень полупрозрачности цвета последнего пиксела.

Все значения, включая и степень полупрозрачности, должны укладываться в диапазон от `0` до `255`. Для степени полупрозрачности значение `0` задает полную прозрачность, а `255` — полную непрозрачность.

Нумерация пикселов в массиве идет слева направо и сверху вниз, т. е. по строкам.

Поскольку набор чисел, хранящийся в свойстве `data` класса `ImageData`, представляет собой коллекцию, для доступа к отдельным значениям мы можем использовать тот же синтаксис, что и в случае обычных массивов. Также мы можем воспользоваться поддерживаемым всеми коллекциями свойством `length`, возвращающим размер коллекции.

Для пустого массива все пикселы будут иметь прозрачный черный цвет. Набор, хранящийся в свойстве `data`, будет содержать нули.

В следующем примере мы задаем для первого пиксела (левого верхнего) полностью непрозрачный красный цвет:

```
var idEmpty = ctxCanvas.createImageData(400, 300);
idEmpty.data[0] = 255;
idEmpty.data[1] = 0;
idEmpty.data[2] = 0;
idEmpty.data[3] = 255;
ctxCanvas.putImageData(idEmpty, 0, 0);
```

Вывод массива пикселов

Завершив формирование нового изображения в массиве пикселов, мы можем вывести его на холст, вызвав метод `putImageData()`:

```
putImageData(<Массив пикселов>, <x1>, <y1>[, <x2>, <y2>,
                                <Ширина>, <Высота>])
```

Второй и третий параметры задают координату точки, где будет находиться левый верхний угол выводимого массива пикселов. Четвертый и пятый параметры задают координаты точки, где находится левый верхний угол фрагмента массива пикселов, который должен быть выведен на холст, а шестой и седьмой — ширину и высоту выводимого фрагмента. Если эти параметры не указаны, будет выведен весь массив пикселов:

```
ctxCanvas.putImageData(idEmpty, 0, 0);
```

В листинге 3.68 приведен код, рисующий на странице прямоугольник с градиентной заливкой. В заливке зеленый цвет плавно сменяется синим, а потом — красным. При этом цвета становятся все более и более прозрачными.

Листинг 3.68. Вывод массива пикселов

```
<!DOCTYPE html>
<html lang="ru">
  <head>
    <meta charset="utf-8">
    <title>Манипуляция пикселами</title>
    <script>
      window.addEventListener("load", function() {
        var oCanvas = document.getElementById("cnv");
        var ctxCanvas = oCanvas.getContext("2d");
```

```
        var idEmpty = ctxCanvas.createImageData(255, 255);
        for (var i = 0; i < idEmpty.data.length; i += 4) {
          idEmpty.data[i + 0] = i / 4 / 255;
          idEmpty.data[i + 1] = 255 - i / 4 / 255;
          idEmpty.data[i + 2] = i / 4 / 255;
          idEmpty.data[i + 3] = 255 - i / 4 / 255;
        }
        ctxCanvas.putImageData(idEmpty, 0, 0);
      }, false);
    </script>
  </head>
  <body>
    <canvas id="cnv" width="400" height="300"></canvas>
  </body>
</html>
```

3.19. Хранилище

Мы уже знаем способ сохранить произвольные данные на стороне клиента — файлы cookies. Однако лучше всего для этой цели пользоваться средствами DOM 3, о которых сейчас пойдет речь.

Хранилище — это инструмент для сохранения на стороне клиента произвольных данных любого типа. Хранилище позволяет запомнить любое значение, указав для него уникальное имя, а впоследствии извлечь его, сославшись на заданное ранее имя, и использовать в вычислениях.

3.19.1. Сессионное и локальное хранилища

Современные Web-браузеры предлагают разработчикам сценариев два различных хранилища:

- *сессионное хранилище* — хранит данные, пока в окне Web-браузера открыта текущая страница. После перехода на другую страницу или закрытия окна все сохраненные данные будут удалены. Сессионное хранилище подходит лишь для временного хранения данных при случайном обновлении страницы посетителем;
- *локальное хранилище* — сохраняет данные даже после перехода на другую страницу или закрытия окна Web-браузера. Позволяет хранить данные неопределенно долгое время.

Нужно помнить, что хранилище любого типа хранит данные, относящиеся к какой-то одной странице. Данные, сохраненные любой другой страницей, при этом считать невозможно — это сделано ради безопасности.

3.19.2. Работа с хранилищем

Сессионное хранилище доступно через свойство sessionStorage, а локальное — через свойство localStorage. Оба этих свойства поддерживаются объектом window.

> **Внимание!**
>
> Хранилища обоих типов доступны лишь в том случае, если страница была загружена с Web-сервера. При загрузке страницы с локального диска свойства sessionStorage и localStorage недоступны.

Хранилище любого типа представляется объектом класса Storage. Все манипуляции по сохранению и чтению данных выполняются посредством вызова методов этого объекта.

В следующем примере мы получаем сессионное хранилище:

```
var stSession = window.sessionStorage;
```

А в этом — получаем доступ к локальному хранилищу с учетом того, что Web-браузер может его не поддерживать:

```
if (window.localStorage) {
   var stLocal = window.localStorage;
   console.log(stLocal);
   // Работаем с локальным хранилищем
}
else {
   console.log("Web-браузер не поддерживает локальное хранилище");
}
```

Метод setItem(<Имя>, <Значение>) выполняет сохранение значения в хранилище. Имя, под которым сохраненное значение можно будет впоследствии отыскать и прочитать, как и само значение, указываются в виде строк. Результат этот метод не возвращает:

```
stLocal.setItem("language", "JavaScript");
```

Для чтения сохраненного ранее значения мы применим метод getItem(<Имя>). Имя, опять же, указывается в виде строки. Метод возвращает извлеченное из хранилища значение в виде строки или null, если значения с указанным именем найти не удалось:

```
var language = stLocal.getItem("language");
if (language) {
   // Используем полученное из хранилища значение
   console.log(language);
}
else {
   console.log("Такого значения в хранилище нет");
}
```

Если понадобится удалить из хранилища сохраненное ранее значение, следует вызвать метод removeItem(<Имя>). Результат он не возвращает:

```
stLocal.removeItem("language");
```

Единственное свойство объекта `Storage` — `length` — возвращает количество сохраненных в хранилище значений в виде числа.

> **Совет**
>
> Посмотреть содержимое хранилища можно на вкладке **Хранилище Инструментов разработчика**. Чтобы отобразить вкладку, переходим в главное меню и выбираем пункт **Разработка | Инспектор хранилища** или нажимаем комбинацию клавиш <Shift>+<F9>.

3.19.3. Использование локального хранилища для временного хранения данных

Локальное хранилище — идеальное средство для временного хранения данных, введенных посетителем в форму. Код из листинга 3.69 демонстрирует его использование для этой цели.

Листинг 3.69. Пример временного хранения данных в хранилище

```
<!DOCTYPE html>
<html lang="ru">
  <head>
    <meta charset="utf-8">
    <title>Хранилище HTML 5</title>
    <script>
      window.addEventListener("load", function() {
        var stLocal = window.localStorage;
        var txtTitle = document.getElementById("txtTitle");
        var txtContent = document.getElementById("txtContent");
        var btnSave = document.getElementById("btnSave");
        var btnLoad = document.getElementById("btnLoad");
        btnLoad.disabled = !((stLocal.getItem("title")) &&
                             (stLocal.getItem("content")));
        btnSave.addEventListener("click", function() {
          if ((txtTitle.value) && (txtContent.value)) {
            stLocal.setItem("title", txtTitle.value);
            stLocal.setItem("content", txtContent.value);
            btnLoad.disabled = false;
          }
        }, false);
        btnLoad.addEventListener("click", function() {
          txtTitle.value = stLocal.getItem("title");
          txtContent.value = stLocal.getItem("content");
        }, false);
      }, false);
    </script>
  </head>
```

```
  <body>
    <form>
      <p>Заголовок статьи<br>
      <input type="text" id="txtTitle"></p>
      <p>Содержимое статьи<br>
      <textarea id="txtContent"></textarea></p>
      <p><input type="submit" value="Отправить"></p>
      <p><input type="button" id="btnSave" value="Сохранить">
      <input type="button" id="btnLoad" value="Загрузить" disabled></p>
    </form>
  </body>
</html>
```

3.20. Средства геолокации

DOM 3 также предоставляет средства для геолокации, т. е. для получения местоположения клиентского компьютера (его географических координат).

3.20.1. Доступ к средствам геолокации

Средства геолокации доступны из свойства с «говорящим» названием `geolocation`, поддерживаемым объектом `navigator`. Они представляют собой объект класса `Geolocation`, с помощью методов которого и выполняется получение местоположения:

```
if (navigator.geolocation) {
   var oGeo = navigator.geolocation;
   // Получаем данные геолокации
} else {
   // Web-браузер не поддерживает геолокацию
}
```

3.20.2. Получение данных геолокации

Проще всего однократно получить данные геолокации, т. е. географические координаты клиентского компьютера. Это можно сделать, обратившись к методу `getCurrentPosition()` класса `Geolocation`:

```
getCurrentPosition(<Функция для получения данных>[,
          <Функция, вызывающаяся в случае ошибки>[, <Параметры>]])
```

Единственным обязательным параметром этому методу передается функция, которая будет вызвана при успешном получении данных геолокации и собственно получит их. В качестве единственного параметра она получит объект класса `Position`, хранящий данные геолокации.

У объекта класса `Position`, полученного функцией, нас интересует свойство `coords`. Оно хранит объект класса `Coordinates`, представляющий координаты клиента и поддерживающий следующие свойства:

- `latitude` — широта в градусах;
- `longitude` — долгота в градусах;
- `altitude` — высота над уровнем моря в метрах. Если значение высоты получить не удается, возвращается `null`;
- `accuracy` — точность определения широты и долготы в метрах;
- `altitudeAccuracy` — точность определения высоты в метрах. Если значение высоты получить не удается, возвращается `null`;
- `heading` — направление движения. Представляет собой угол в градусах, отмеренный от направления на север по часовой стрелке. Если клиентский компьютер никуда не движется, возвращает `NaN`, если значение направления получить не удается, — `null`;
- `speed` — скорость движения в метрах в секунду. Если клиентский компьютер никуда не движется, возвращает `NaN`. Если значение скорости получить не удается, возвращается `null`.

Все эти свойства возвращают числовые значения.

Пример получения данных геолокации приведен в листинге 3.70.

Листинг 3.70. Получение данных геолокации

```
if (navigator.geolocation) {
  navigator.geolocation.getCurrentPosition(function(pos) {
    var lat = pos.coords.latitude;
    var lon = pos.coords.longitude;
    var alt = pos.coords.altitude;
    var heading = pos.coords.heading;
    if (isNaN(heading)) heading = 0;
    var speed = pos.coords.speed;
  });
}
```

3.20.3. Обработка нештатных ситуаций

При получении данных геолокации возможны нештатные ситуации — например, ошибки в работе инструментов определения местоположения, встроенных в клиентский компьютер. При их возникновении будет выполнена функция, переданная методу `getCurrentPosition()` вторым параметром.

В качестве единственного параметра эта функция получит объект класса `PositionError`, хранящий сведения о возникшей ошибке. Этот класс поддерживает два свойства:

- `code` — числовой код ошибки:
  - `1` — посетитель запретил странице доступ к данным геолокации;
  - `2` — в работе инструментов для определения местоположения, встроенных в компьютер, возникла ошибка;
  - `3` — истекло время, отведенное на определение местоположения;
- `message` — текстовое описание ошибки.

Пример обработки нештатных ситуаций приведен в листинге 3.71.

Листинг 3.71. Обработка нештатных ситуаций

```
navigator.geolocation.getCurrentPosition(function(pos) {
  // Получаем данные геолокации
}, function(error) {
  switch (error.code) {
    case 1:
      window.alert("Нет доступа к данным геолокации");
      break;
    case 2:
      window.alert("Внутренняя ошибка");
      break;
    case 3:
      window.alert("Истекло время, отведенное на получение данных");
      break;
  }
});
```

3.20.4. Задание дополнительных параметров

При получении сведений о местоположении можно задать дополнительные параметры. Они указываются третьим параметром метода `getCurrentPosition()` в виде объекта, поддерживающего указанные далее свойства:

- `enableHighAccuracy` — задает точность получения данных: если `true`, то устройство будет по возможности получать максимально точные данные геолокации, если `false` — данные меньшей точности. Следует помнить, что получение точных данных может потребовать больше времени и повысить энергопотребление устройства. Значение по умолчанию — `false`;
- `maximumAge` — указывает время (в виде числа в миллисекундах), в течение которого полученные данные геолокации будут кэшироваться устройством. Если задано `0`, устройство не будет кэшировать данные, если `Infinity` — эти данные будут кэшироваться до тех пор, пока не устареют (пока компьютер не изменит свое местоположение). Значение по умолчанию — `Infinity`;
- `timeout` — указывает время (в виде числа в миллисекундах), отведенное на получение сведений о местоположении. По истечении этого времени будет сге-

нерирована ошибка, которую можно отследить. Если задано `Infinity`, то время ограничиваться не будет. Значение по умолчанию — `Infinity`.

Пример указания дополнительных параметров приведен в листинге 3.72.

Листинг 3.72. Пример указания дополнительных параметров `getCurrentPosition()`

```
navigator.geolocation.getCurrentPosition(function(pos) {
   // Получаем данные геолокации
}, function(error) {
   // Отслеживаем нештатные ситуации
}, {
   enableHighAccuracy: true,
   maximumAge: 10000,
   timeout: 5000
});
```

Здесь мы задаем получение точных координат и кэширование их в течение 10 секунд, а также отводим на их получение время, равное 5 секундам.

3.20.5. Отслеживание местоположения компьютера

Наконец, мы можем постоянно отслеживать местоположение клиентского компьютера, чтобы выяснить, куда он перемещается и достиг ли он точки назначения. Для этого нужно использовать метод `watchPosition()`, формат вызова которого совпадает с таковым у метода `getCurrentPosition()`.

Как только местоположение клиентского компьютера изменится, будет вызвана функция, переданная этому методу первым параметром. А при возникновении нештатной ситуации — функция, переданная вторым параметром.

Метод `watchPosition()` возвращает особый идентификатор, который позволяет отменить отслеживание местоположения. Это можно сделать вызовом не возвращающего результат метода `clearWatch()`:

```
<Объект геолокации>.clearWatch(<Идентификатор>)
```

В листинге 3.73 показан код, отслеживающий местоположение клиента, чтобы выяснить, достиг ли он точки назначения. Координаты точки назначения хранятся в переменных `latD` (широта) и `lonD` (долгота).

Листинг 3.73. Пример отслеживания местоположения компьютера

```
var wp = navigator.geolocation.watchPosition(function(pos) {
  if ((latD == pos.coords.latitude) && (lonD == pos.coords.longitude)) {
    navigator.geolocation.clearWatch(wp);
    window.alert("Вы в точке назначения!");
  }
```

```
}, function(error) {
  window.alert(error.message);
});
```

На этом мы заканчиваем знакомство с клиентскими технологиями и переходим к изучению технологий, которые выполняются на стороне сервера. Но вначале на компьютер необходимо установить специальное программное обеспечение. Какое программное обеспечение необходимо, где его найти и как установить, мы рассмотрим в следующей главе.

ГЛАВА 4



Программное обеспечение Web-сервера. Устанавливаем и настраиваем программы под Windows

4.1. Необходимые программы

Для тестирования программ необходимо установить на компьютер специальное программное обеспечение:

- *Web-сервер Apache* — программное обеспечение, отвечающее за отображение документов, запрашиваемых при наборе URL-адреса в адресной строке Web-браузера;
- *Интерпретатор PHP* — для выполнения программ, написанных на языке PHP;
- *MySQL* — сервер баз данных;
- *phpMyAdmin* — набор скриптов на PHP для управления базами данных.

Все эти программы можно бесплатно получить с сайтов производителей.

Необходимо сразу заметить, что программное обеспечение мы устанавливаем только для тестирования и не задаемся целью охватить все его настройки.

> ***Внимание!***
>
> В последних версиях Windows при внесении в систему ключевых изменений (задание настроек, установка программ и т. п.) выводится на экран предупреждение подсистемы UAC. На такие предупреждения всегда следует отвечать положительно, в противном случае изменения не будут внесены, и программы могут оказаться неработоспособными.

> ***Примечание***
>
> В приведенных далее инструкциях по установке указываются точные версии устанавливаемых программ. Скорее всего, ко времени выхода книги из печати будут выпущены новые версии. В этом случае рекомендуется использовать их, особенно если номера версий различаются только последними цифрами. Вероятно, процесс установки новых версий программ будет мало отличаться от описанного в книге, однако следует иметь в виду, что незначительные различия все же могут присутствовать.

Прежде чем устанавливать программы, необходимо проверить сетевые настройки и убедиться в отсутствии программ, занимающих порты 80 и 3306, т. к. эти порты используют Web-сервер Apache и сервер MySQL. Для проверки запустим командную строку Windows. Вполне возможно, что вы никогда не пользовались командной строкой и не знаете, как она запускается. Давайте рассмотрим некоторые способы ее запуска в Windows:

- через поиск находим приложение **Командная строка**;
- нажимаем комбинацию клавиш <Windows>+<R>. В открывшемся окне вводим `cmd` и нажимаем кнопку **OK**;
- находим файл cmd.exe в папке C:\Windows\System32;
- в Проводнике щелкаем правой кнопкой мыши на свободном месте списка файлов, удерживая при этом нажатой клавишу <Shift>. Из контекстного меню выбираем пункт **Открыть окно команд**.

В командной строке набираем команду:

```
ping 127.0.0.1
```

Если число потерянных пакетов больше 0, то необходимо проверить сетевые настройки. Чтобы проверить порты 80 и 3306, в командной строке набираем команду:

```
netstat -anb
```

В полученном списке не должно быть строк с портами 80 и 3306. Если они есть, то Apache или MySQL не смогут запуститься. Обычно эти порты занимают программы Skype и Web-сервер IIS. Перед установкой и использованием Apache и MySQL запускать эти программы не следует.

> ***Внимание!***
>
> Для получения результатов выполнения команды `netstat` необходимо запустить командную строку с правами администратора. Чтобы это сделать, через поиск находим приложение **Командная строка**, щелкаем на его значке правой кнопкой мыши и выбираем команду **Запуск от имени администратора**.

4.2. Установка XAMPP

Все необходимые нам программы входят в пакет XAMPP, который мы сейчас и установим. Для этого переходим на сайт **https://www.apachefriends.org/** и загружаем файл xampp-win32-7.2.0-0-VC15-installer.exe или более позднюю его версию. Запускаем программу установки и в открывшемся окне (рис. 4.1) нажимаем кнопку **Next**. На следующем шаге (рис. 4.2) проверяем установку всех флажков и нажимаем кнопку **Next**. Далее выбираем место установки (рис. 4.3) и нажимаем кнопку **Next**. Лучше всего установить пакет в папку C:\xampp, которая используется по умолчанию. На следующих двух шагах (рис. 4.4 и 4.5) тоже нажимаем кнопку **Next**. В процессе установки будет выведено окно с запросом блокировки запуска сервера Apache. Следует в обязательном порядке разрешить запуск, иначе ничего работать не станет. На последнем шаге (рис. 4.6) нажимаем кнопку **Finish**.

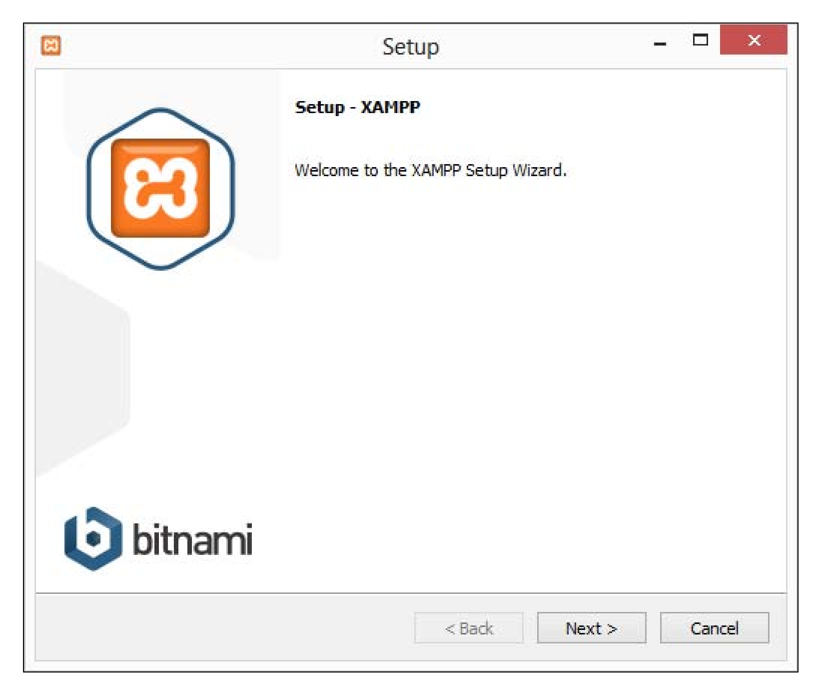


Рис. 4.1. Установка XAMPP: шаг 1

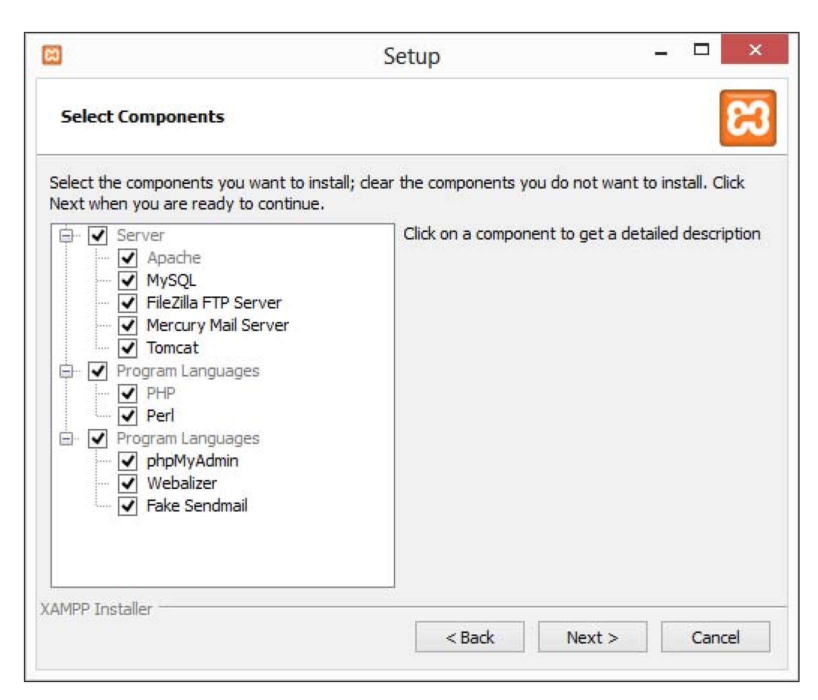


Рис. 4.2. Установка XAMPP: шаг 2

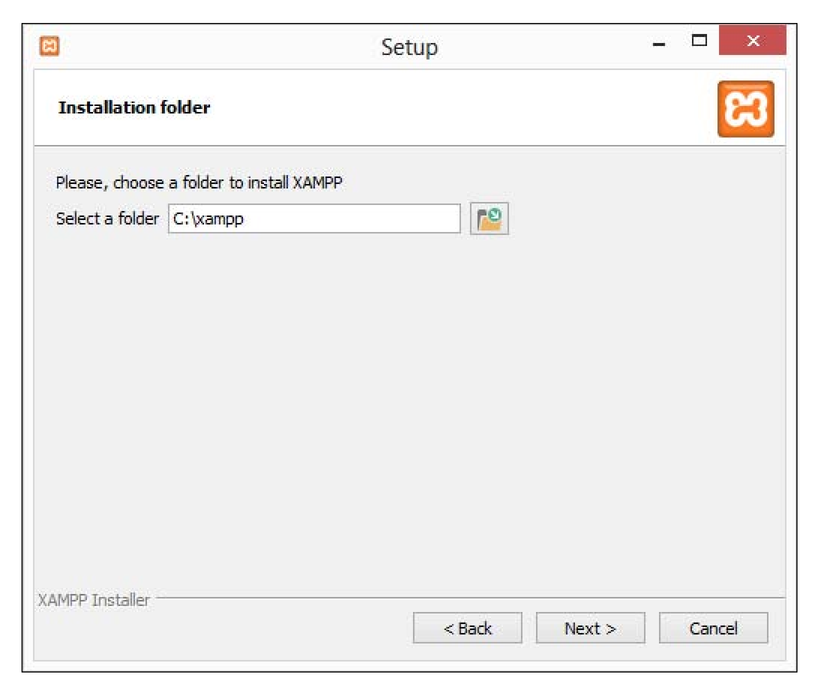


Рис. 4.3. Установка XAMPP: шаг 3

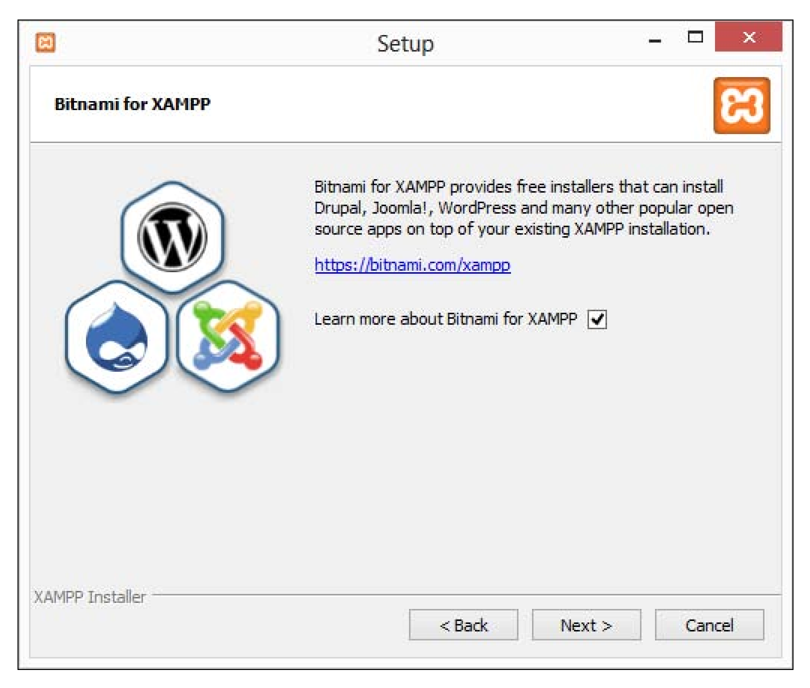


Рис. 4.4. Установка XAMPP: шаг 4

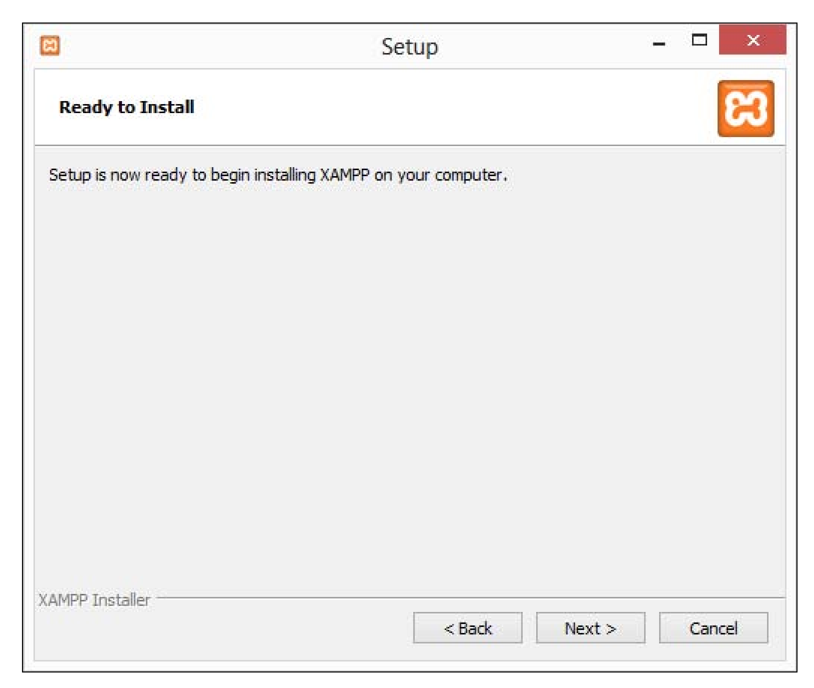


Рис. 4.5. Установка XAMPP: шаг 5



Рис. 4.6. Установка XAMPP: шаг 6

Если на последнем шаге флажок, имеющийся в окне, показанном на рис. 4.6, не был сброшен, то автоматически запустится приложение XAMPP Control Panel. В случае, если приложение не запустилось, переходим в папку C:\xampp и запускаем на выполнение файл xampp-control.exe. В результате отобразится окно, показанное на рис. 4.7.

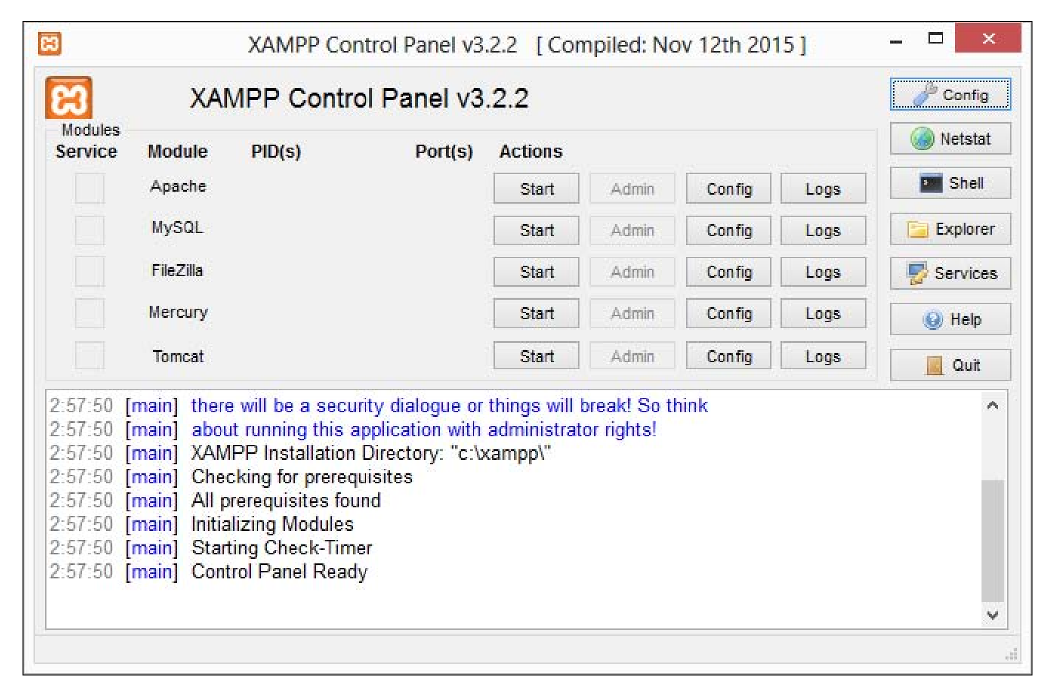


Рис. 4.7. Окно **XAMPP Control Panel**

Давайте попробуем запустить серверы. Для этого нажимаем кнопки **Start** у пунктов **Apache** и **MySQL**. При успешном запуске названия серверов будут выделены зеленым цветом, а справа отобразятся идентификаторы процессов и номера портов, на которых серверы запущены (рис. 4.8). При этом кнопки **Start** будут заменены на кнопки **Stop**, с помощью которых можно будет серверы остановить.

> ***Примечание***
>
> Если при попытке запуска серверов возникла ошибка, то вначале нужно проверить, а не блокирует ли программы брандмауэр Windows или антивирусная программа. Если добавление серверов в список их исключений проблему не решает, то, вполне возможно, на компьютере не хватает библиотек динамической компоновки. В этом случае следует установить исполняемую среду Microsoft Visual C++. Она доступна бесплатно на сайте Microsoft.

После нажатия красной кнопки с крестиком в заголовке окна **XAMPP Control Panel** окно не закрывается, а просто скрывается. Значок приложения доступен на панели задач Windows. Если щелкнуть на этом значке правой кнопкой мыши, то отобразится контекстное меню (рис. 4.9), с помощью которого можно открыть окно **XAMPP Control Panel** или управлять работой серверов.

XAMPP Control Panel v3.2.2 [Compiled: Nov 12th 2015]

XAMPP Control Panel v3.2.2

Modules

Service	Module	PID(s)	Port(s)	Actions			
	Apache	6504 10728	80, 443	Stop	Admin	Config	Logs
	MySQL	6728	3306	Stop	Admin	Config	Logs
	FileZilla			Start	Admin	Config	Logs
	Mercury			Start	Admin	Config	Logs
	Tomcat			Start	Admin	Config	Logs

Config
Netstat
Shell
Explorer
Services
Help
Quit

```
2:57:50 [main]   All prerequisites found
2:57:50 [main]   Initializing Modules
2:57:50 [main]   Starting Check-Timer
2:57:50 [main]   Control Panel Ready
2:58:22 [Apache]        Attempting to start Apache app...
2:58:22 [Apache]        Status change detected: running
2:58:30 [mysql]         Attempting to start MySQL app...
2:58:30 [mysql]         Status change detected: running
```

Рис. 4.8. Окно **XAMPP Control Panel**: серверы Apache и MySQL успешно запущены

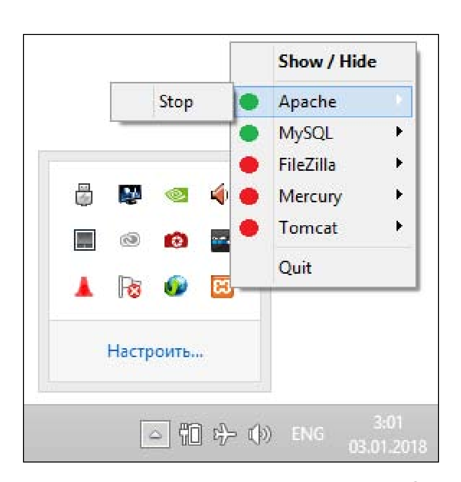


Рис. 4.9. Контекстное меню приложения **XAMPP Control Panel**

Для проверки работоспособности Web-сервера Apache открываем Web-браузер и в адресной строке вводим: `http://localhost/`. В результате должно отобразиться приветствие пакета XAMPP (рис. 4.10). Чтобы проверить работоспособность сервера MySQL и программы phpMyAdmin, в адресной строке Web-браузера набираем: `http://localhost/phpmyadmin/`. Результат успешной проверки показан на рис. 4.11.

Успешный запуск программы phpMyAdmin уже свидетельствует о работоспособности интерпретатора PHP, но мы попробуем получить полную информацию об интерпретаторе и заодно научимся запускать программы. Для этого потребуется

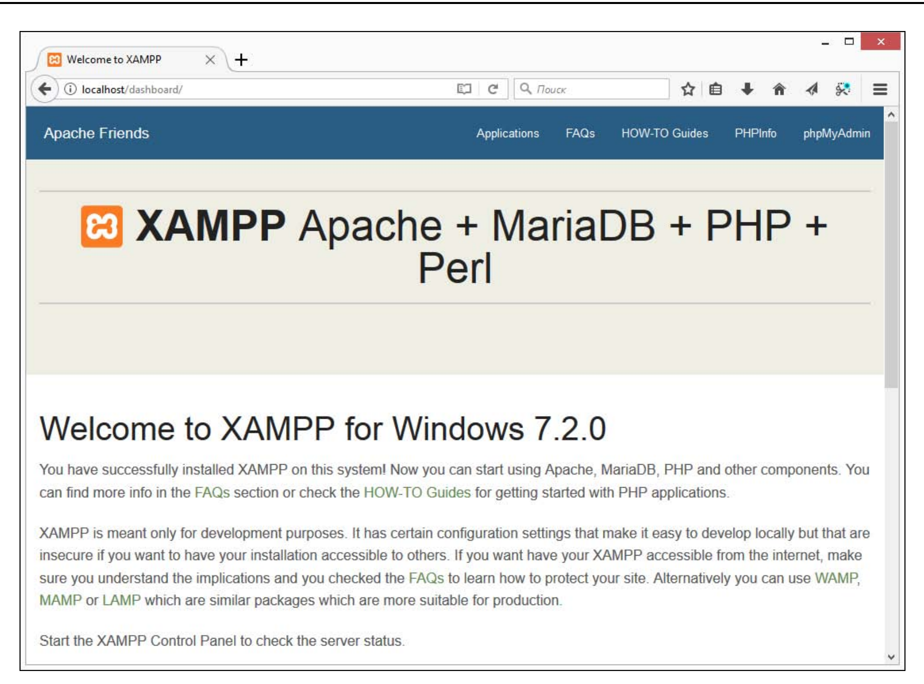


Рис. 4.10. Приветствие пакета XAMPP

Рис. 4.11. Программа phpMyAdmin

изменить содержимое файла index.php, расположенного в папке C:\xampp\htdocs. Открываем файл в программе Notepad++, удаляем содержимое, а затем вводим код из листинга 4.1.

Листинг 4.1. Вывод полной информации об интерпретаторе PHP

```
<?php
phpinfo();
?>
```

Сохраняем файл, переходим в Web-браузер и в адресной строке вводим: `http://localhost/`. Результат успешного выполнения программы показан на рис. 4.12.

phpinfo()

localhost

Поиск

PHP Version 7.2.0

php

System	Windows NT ASUS 6.2 build 9200 (Windows 8 Home Premium Edition) i586
Build Date	Nov 28 2017 23:47:42
Compiler	MSVC15 (Visual C++ 2017)
Architecture	x86
Configure Command	cscript /nologo configure.js "--enable-snapshot-build" "--enable-debug-pack" "--with-pdo-oci=c:\php-snap-build\deps_aux\oracle\x86\instantclient_12_1\sdk,shared" "--with-oci8-12c=c:\php-snap-build\deps_aux\oracle\x86\instantclient_12_1\sdk,shared" "--enable-object-out-dir=../obj/" "--enable-com-dotnet=shared" "--without-analyzer" "--with-pgo"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\Windows
Loaded Configuration File	C:\xampp\php\php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20170718
PHP Extension	20170718
Zend Extension	320170718
Zend Extension Build	API320170718,TS,VC15
PHP Extension Build	API20170718,TS,VC15
Debug Build	no
Thread Safety	enabled
Zend Signal Handling	disabled
Zend Memory Manager	enabled

Рис. 4.12. Результат выполнения функции `phpinfo()`

Помимо PHP, пакет XAMPP содержит поддержку языка программирования Perl. Программы, написанные на языке Perl, нужно размещать в папке C:\xampp\cgi-bin, а не в папке C:\xampp\htdocs. После установки пакета эта папка содержит несколько тестовых программ. Давайте проверим работоспособность Perl. Переходим в Web-браузер и в адресной строке вводим: `http://localhost/cgi-bin/perltest.cgi`. Если на Web-странице отобразился заголовок **GCI with MiniPerl**, то все работает правильно.

4.3. Структура каталогов сервера Apache

Итак, сервер установлен и запущен. Теперь давайте рассмотрим каталоги сервера Apache, их содержание и назначение. В папке C:\xampp\apache находятся следующие каталоги:

- bin — здесь располагается главный исполняемый файл сервера (httpd.exe) и исполняемые файлы вспомогательных утилит;
- conf — каталог, где находятся конфигурационный файл сервера (httpd.conf) и другие файлы конфигурации сервера Apache;
- error — каталог для файлов с сообщениями об ошибках (например, если запрашиваемый файл не найден);
- icons — здесь содержится ряд изображений, используемых в листингах каталогов;
- include — набор заголовочных файлов, необходимых для разработки дополнительных модулей (нам они не понадобятся);
- lib — набор библиотечных файлов, предназначенных для разработки дополнительных модулей (они также нам не нужны);
- logs — в этой папке находятся журналы регистрации посещений (access.log) и ошибок (error.log), позволяющие получить подробную информацию обо всех запросах и ошибках. Открыть эти файлы можно с помощью любого текстового редактора (например, Notepad++);
- manual — здесь находятся файлы документации. Просматривать документацию следует не в этом каталоге, а набрав в командной строке Web-браузера: `http://localhost/manual/`. Не пытайтесь набрать этот адрес прямо сейчас. Документация по этому адресу будет доступна после того, как мы внесем изменения в конфигурационный файл. Для этого нужно в конфигурационном файле httpd.conf (C:\xampp\apache\conf\httpd.conf) убрать символ комментария (`#`) перед строкой:

  ```
  #Include conf/extra/httpd-manual.conf
  ```

  и перезагрузить сервер;
- modules — этот каталог содержит подключаемые модули.

Следующие каталоги, расположенные в папке C:\xampp, также относятся к серверу Apache:

- htdocs — здесь должны располагаться файлы в форматах HTML и PHP, а также другие файлы, которые будут доступны при наборе `http://localhost/` в адресной строке Web-браузера (например, изображения, файлы каскадной таблицы стилей и т. д.).

  С этим каталогом мы будем работать постоянно. Поэтому удобно добавить ярлык к нему на рабочий стол. Для этого щелкаем на названии каталога правой кнопкой мыши, в контекстном меню выбираем пункт **Отправить** и в появившемся подменю выбираем пункт **Рабочий стол (создать ярлык)**;

- cgi-bin — каталог для CGI-программ (программ, написанных на языках Perl, C и т. д.).

После установки сервера в каталоге htdocs находится приветствие пакета XAMPP, которое мы видим при наборе `http://localhost/` в адресной строке Web-браузера. Файл по умолчанию, содержимое которого будет отправлено Web-браузеру, должен иметь название index.php, index.html и т. д. Полный список файлов по умолчанию определяет директива `DirectoryIndex` в конфигурационном файле httpd.conf. Причем порядок указания названий файлов задает приоритет: если вначале указан файл index.php, а затем index.html, то при наличии двух файлов будет запущен файл index.php. Чтобы в этом случае получить содержимое файла index.html, нужно его название указать явным образом: `http://localhost/index.html`.

В пакете XAMPP по умолчанию для PHP задана кодировка UTF-8, а для HTML-документов кодировка по умолчанию не указана. Давайте это исправим. Редактировать конфигурационный файл httpd.conf мы не станем, а просто добавим в каталог C:\xampp\htdocs файл .htaccess. Этот файл позволяет изменять значения некоторых директив для каталога, в котором он расположен, и для всех вложенных подкаталогов. Обратите внимание: у файла нет имени, только точка и расширение. В файл .htaccess добавим следующие директивы:

```
DefaultLanguage ru
AddDefaultCharset UTF-8
```

Файл .htaccess обрабатывается при каждом запросе файла, тогда как конфигурационный файл httpd.conf обрабатывается только один раз при запуске сервера Apache. Поэтому при изменении файла .htaccess перезагружать сервер нет необходимости, а при изменении файла httpd.conf это нужно делать в обязательном порядке. На виртуальном хостинге можно будет использовать только файл .htaccess.

При изучении PHP мы будем иногда работать с кодировкой windows-1251. Так как по умолчанию для PHP задана кодировка UTF-8, то при запуске программы без явного указания кодировки мы получим знаки вопроса вместо русских букв. Чтобы этого избежать, в каталоге C:\xampp\htdocs создадим два каталога: cp1251 и utf8. В каталоге cp1251 будем хранить файлы в кодировке windows-1251, а в каталоге utf8 — в кодировке UTF-8. В каталог cp1251 добавляем файл .htaccess со следующими директивами:

```
DefaultLanguage ru
AddDefaultCharset windows-1251
php_value default_charset "windows-1251"
```

В каталог utf8 файл .htaccess добавлять не нужно, т. к. кодировка для PHP задана в конфигурационном файле php.ini (он расположен в каталоге C:\xampp\php), а кодировка по умолчанию для HTML-документов будет взята из файла .htaccess, расположенного в каталоге C:\xampp\htdocs.

Для проверки правильности настройки в каталогах cp1251 и utf8 создаем файлы с названием index.php. Внутрь файлов добавляем следующий код:

```
<?php
echo 'Русский текст';
?>
```

При сохранении файлов обратите внимание на кодировку файла с программой. Чтобы увидеть кодировку файла в редакторе Notepad++, открываем меню **Кодировки**. Для файла index.php, расположенного в каталоге C:\xampp\htdocs\cp1251, должен быть установлен флажок **Кодировки | Кириллица | Windows-1251**, а для файла index.php, расположенного в папке C:\xampp\htdocs\utf8, — флажок **Кодировать в UTF-8 (без BOM)**. Открываем Web-браузер и в адресной строке вначале вводим: `http://localhost/cp1251/`, а затем: `http://localhost/utf8/`. Русский текст в обоих случаях должен отображаться правильно.

> ***Внимание!***
>
> При удалении пакета XAMPP выводится диалоговое окно с запросом удаления папки *htdocs*. Если в этом окне выбрать кнопку **OK**, то все проекты будут удалены без возможности восстановления. Будьте очень внимательны!

4.4. Файл конфигурации Apache (httpd.conf)

Файл httpd.conf (C:\xampp\apache\conf\httpd.conf) — это основной файл конфигурации сервера Apache. Открыть и отредактировать этот файл можно с помощью любого текстового редактора, например Блокнота или Notepad++. После каждого изменения в файле конфигурации необходимо перезагрузить сервер, остановив его и запустив снова. До перезагрузки он будет работать со старыми параметрами.

Внутри файла httpd.conf с помощью директивы `Include` могут подключаться дополнительные конфигурационные файлы (они расположены в каталоге C:\xampp\apache\conf\extra):

```
Include conf/extra/httpd-languages.conf
```

4.4.1. Основные понятия

В файле httpd.conf содержатся *директивы*, влияющие на работу сервера Apache. Директива представляет собой ключевое слово, за которым следует одно или несколько значений. Директивы бывают простыми (изменяющими только одно свойство сервера), а могут объединяться в разделы (позволяют изменять сразу несколько свойств какого-нибудь объекта).

Если в начале строки указан символ `#`, то такая строка является комментарием:

```
# ServerAdmin: Your address, where problems with the server should be
# e-mailed.  This address appears on some server-generated pages, such
# as error documents.  e.g. admin@your-domain.com
ServerAdmin postmaster@localhost
```

В этом примере первые три строки закомментированы, а четвертая с помощью директивы `ServerAdmin` задает электронный адрес администратора сервера.

4.4.2. Разделы файла конфигурации

Директивы могут объединяться в разделы, что позволяет ограничить область действия директив отдельным каталогом, набором файлов или набором URL. Существуют следующие разделы:

- `Directory` и `DirectoryMatch` — указывают, что директивы применимы к заданному каталогу и всем подкаталогам:

```
<Directory "C:/xampp/htdocs">
   Options Indexes FollowSymLinks Includes ExecCGI
   AllowOverride All
   Require all granted
</Directory>
```

  Раздел `DirectoryMatch` позволяет использовать регулярные выражения;

- `Files` и `FilesMatch` — указывают, что директивы применимы только к определенным файлам. Символ `*` соответствует любой последовательности символов, а символ `?` — любому одиночному символу. В качестве примера запретим доступ к файлам .htaccess и .htpasswd:

```
<Files ".ht*">
   Require all denied
</Files>
```

  Раздел `FilesMatch` позволяет использовать регулярные выражения;

- `IfModule` — указывает, что директивы будут задействованы лишь при загрузке указанного модуля:

```
<IfModule dir_module>
   DirectoryIndex index.html
</IfModule>
```

- `Limit` и `LimitExcept`. `Limit` — указывают, что директивы будут использоваться, только когда HTTP-запрос выполнен с помощью одного из указанных методов (`GET`, `POST` и др.). `LimitExcept` задает директивы, которые будет применены для методов, не указанных в списке аргументов:

```
<Limit GET POST OPTIONS PROPFIND>
   Require all granted
</Limit>
```

- `Location` и `LocationMatch` — определяют, что заключенные в них директивы действуют лишь в случае обращения с указанного интернет-адреса:

```
<Location /server-status>
   SetHandler server-status
</Location>
```

  Раздел `LocationMatch` позволяет использовать регулярные выражения;

- `VirtualHost` — определяет, что директивы применимы только к документам указанного виртуального хоста. Применяется, когда сервер обслуживает множество Web-сайтов с разными именами хостов:

```
<VirtualHost 192.168.0.1:80>
   ServerAdmin webmaster@site.ru
   DocumentRoot /www/docs/site.ru
   ServerName site.ru
</VirtualHost>
```

4.4.3. Общие директивы. Создание домашнего каталога пользователя, доступного при запросе `http://localhost/~nik/`

Приведем основные общие директивы сервера Apache:

- `ServerName` — определяет имя сервера:

  ```
  ServerName localhost:80
  ```

- `ServerAdmin` — задает E-mail администратора сервера:

  ```
  ServerAdmin postmaster@localhost
  ```

- `ServerRoot` — указывает местонахождение каталогов сервера:

  ```
  ServerRoot "C:/xampp/apache"
  ```

- `DocumentRoot` — определяет местонахождение корневого каталога для документов на сервере:

  ```
  DocumentRoot "C:/xampp/htdocs"
  ```

- `UserDir` — задает имя каталога, в котором ищутся домашние каталоги пользователей при получении запроса вроде **http://localhost/~user/**:

  ```
  UserDir "C:/xampp/apache/user"
  ```

  Создадим каталог для пользователя `nik`. Для этого добавим в каталог C:\xampp\apache каталог user и создадим в нем каталог nik, в который добавим файл index.html со следующим содержанием:

  ```
  <!DOCTYPE html>
  <html lang="ru">
  <head>
     <meta charset="utf-8">
     <title>Страничка пользователя Nik</title>
  </head>
  <body>Привет всем</body>
  </html>
  ```

  Далее с помощью Notepad++ открываем файл httpd-userdir.conf (который находится в каталоге C:\xampp\apache\conf\extra), удаляем его содержимое и вставляем следующие директивы:

  ```
  <IfModule userdir_module>
     UserDir "C:/xampp/apache/user"
     <Directory "C:/xampp/apache/user">
  ```

```
        AllowOverride All
        Options MultiViews Indexes IncludesNoExec
        <Limit GET POST OPTIONS>
           Require all granted
        </Limit>
        <LimitExcept GET POST OPTIONS>
           Require all denied
        </LimitExcept>
    </Directory>
</IfModule>
```

Сохраняем и закрываем файл. Теперь файл httpd-userdir.conf необходимо подключить к основному конфигурационному файлу. Открываем файл httpd.conf и убираем символ комментария (#) перед строками:

```
#Include conf/extra/httpd-userdir.conf
#LoadModule userdir_module modules/mod_userdir.so
```

Сохраняем и закрываем файл httpd.conf. Перезапускаем сервер Apache. Далее открываем Web-браузер и в адресной строке набираем: `http://localhost/~nik/`. В итоге в окне Web-браузера должна отобразиться надпись **Привет всем**;

- `PidFile` — указывает местоположение файла, в котором будет регистрироваться исходный процесс сервера:

  ```
  PidFile "logs/httpd.pid"
  ```

- `Listen` — связывает Apache с определенным портом и (или) IP-адресом:

  ```
  Listen 80
  Listen 12.34.56.78:80
  ```

- `Options` — позволяет включить или отключить те или иные опции в различных частях сайта. Если опция помечена знаком `+`, то она добавляется к числу уже включенных опций, а если знаком `-`, то опция отключается. Могут быть заданы следующие опции:

  - `All` — включает все опции, кроме `MultiViews`:

    ```
    Options All
    ```

  - `None` — отключает все опции, кроме `MultiViews`:

    ```
    Options None
    ```

  - `ExecCGI` — позволяет выполнять CGI-программы в каталоге, отличном от указанного в директиве `ScriptAlias`, — например, в каталоге с обычными документами. Для правильной работы необходимо указать директиву `AddHandler` или `SetHandler`:

    ```
    <Directory "C:/xampp/htdocs">
       Options +ExecCGI
       SetHandler cgi-script
    </Directory>
    ```

* `FollowSymLinks` — разрешает использование символических ссылок:

  ```
  Options +FollowSymLinks
  ```

* `SymLinksIfOwnerMatch` — разрешает символические ссылки, если ссылка указывает на объект, который принадлежит тому же пользователю, что и ссылка:

  ```
  Options +SymLinksIfOwnerMatch
  ```

* `Includes` — разрешает использование серверных расширений (SSI):

  ```
  Options +Includes
  ```

* `IncludesNOEXEC` — разрешает серверные расширения, но запрещает команду `#exec` и директиву `#include` для загрузки CGI-программ:

  ```
  Options +IncludesNOEXEC
  ```

* `Indexes` — если эта опция включена и заданный по умолчанию файл не найден, то сервер генерирует листинг файлов. Если опция выключена, то вместо листинга отображается сообщение об ошибке 403:

  ```
  <Directory "C:/xampp/htdocs">
     Options -Indexes
  </Directory>
  ```

  На виртуальном хостинге эта опция должна быть обязательно выключена, иначе пользователь будет видеть все содержимое каталога, в том числе и файлы паролей;

* `MultiViews` — включает content-соответствие — средство, с помощью которого сервер определяет, какой документ наиболее приемлем для посетителя:

  ```
  Options +MultiViews
  ```

4.4.4. Переменные сервера и их использование

Очень часто бывает необходимо записать какое-либо значение, например путь к папке, сразу в нескольких местах файла конфигурации. В таком случае мы можем объявить *переменную сервера*, сохранить в ней это значение, после чего обратиться к этой переменной по ее имени.

Для объявления переменной сервера предусмотрена директива `Define`:

```
Define <Имя переменной> <Значение>
```

В имени переменной допустимы лишь символы латиницы и цифры, причем начинаться имя должно с буквы. Обычно имена переменных набирают прописными буквами.

Создадим переменную `SRVROOT` и присвоим ей путь к папке, где установлен сервер Apache:

```
Define SRVROOT "C:/xampp/apache"
```

Для доступа к переменной используется следующий формат:

```
${<имя переменной>}
```

Примеры:

```
ServerRoot "${SRVROOT}"
<Directory "${SRVROOT}/site1">
</Directory>
```

Выполнить какие-либо директивы, если переменная определена (или не определена — перед именем переменной в этом случае указывается символ `!`), позволяет директива `IfDefine`:

```
<IfDefine SRVROOT>
   ServerRoot "${SRVROOT}"
</IfDefine>
<IfDefine !SRVROOT>
   ServerRoot "C:/xampp/apache"
</IfDefine>
```

4.4.5. Директивы управления производительностью

При увеличении нагрузки на сервер создаются новые процессы, а при уменьшении — эти процессы закрываются. Частые запуски и остановки порожденных процессов снижают производительность сервера, поэтому необходимо правильно настроить следующие директивы:

- `StartServers` — число копий процесса сервера, которые будут созданы при запуске сервера;
- `MinSpareServers` — минимальное число порожденных процессов;
- `MaxSpareServers` — максимальное число порожденных процессов;
- `MaxRequestWorkers` — максимальное число возможных подключений к серверу.

Указанные директивы не применимы к платформе Windows. Их нужно заменять на `StartThreads`, `MinSpareThreads`, `MaxSpareThreads` и `MaxThreads`. Также применяются следующие директивы:

- `ThreadsPerChild` — задает максимальное число потоков, порождаемых каждым дочерним процессом сервера Apache:

  ```
  ThreadsPerChild 150
  ```

- `MaxConnectionsPerChild` — определяет, сколько запросов может обработать порожденный процесс за время его существования. Для снятия ограничений необходимо указать `0`. На платформе Windows директива всегда должна иметь значение `0`:

  ```
  MaxConnectionsPerChild 0
  ```

4.4.6. Директивы обеспечения постоянного соединения

За обеспечение постоянного соединения отвечают следующие директивы:

- `Timeout` — задает промежуток времени в секундах, в течение которого сервер продолжает попытки возобновления приостановленной передачи данных:

  ```
  Timeout 300
  ```

- `KeepAlive` — разрешает постоянные соединения:

  ```
  KeepAlive On
  ```

- `MaxKeepAliveRequests` — ограничивает число допустимых запросов на одно соединение:

  ```
  MaxKeepAliveRequests 100
  ```

  Для снятия ограничений необходимо указать `0`;

- `KeepAliveTimeout` — определяет тайм-аут для постоянного соединения:

  ```
  KeepAliveTimeout 5
  ```

4.4.7. Директивы работы с языками

Для работы с языками предназначены следующие директивы:

- `AddDefaultCharset` — указывает кодовую таблицу для документов по умолчанию:

  ```
  AddDefaultCharset UTF-8
  ```

- `AddCharset` — устанавливает взаимосвязь между кодовой таблицей символов и расширением файла:

  ```
  AddCharset ISO-2022-JP .jis
  ```

- `RemoveCharset` — удаляет взаимосвязь между кодовой таблицей символов и расширением файла;
- `AddLanguage` — устанавливает взаимосвязь между языком и расширением файла:

  ```
  AddLanguage ru .ru
  ```

- `RemoveLanguage` — удаляет все взаимосвязи между языками и расширениями файла;
- `DefaultLanguage` — определяет, какой язык должен быть указан в заголовке, если для расширения файла не задан определенный язык:

  ```
  DefaultLanguage ru
  ```

- `LanguagePriority` — задает приоритет различных языков:

  ```
  LanguagePriority ru en ca cs da de el
  ```

4.4.8. Директивы перенаправления

Приведем основные директивы перенаправления:

- `Alias` и `AliasMatch` — позволяют предоставить доступ не только к файлам, находящимся в каталоге, указанном в директиве `DocumentRoot`, но и к другим каталогам сервера. В директиве `AliasMatch` можно использовать регулярные выражения:

```
AliasMatch ^/manual(?:/(?:de|en|es|ru))?(/.*)?$
"C:/xampp/apache/manual$1"
```

- `ScriptAlias` и `ScriptAliasMatch` — задают местоположение каталога для CGI-сценариев:

```
ScriptAlias /cgi-bin/ "C:/xampp/cgi-bin/"
```

  Директива `ScriptAliasMatch` позволяет использовать регулярные выражения;

- `Redirect` и `RedirectMatch` — сообщают, что искомый документ больше не находится в данном месте, и указывают, где можно его найти. Имеют дополнительный параметр, указывающий состояние переадресации, который может принимать следующие значения:
  - `permanent` — ресурс перемещен навсегда (код 301);
  - `temp` — ресурс перемещен временно (код 302);
  - `seeother` — ресурс был заменен другим ресурсом (код 303);
  - `gone` — ресурс удален навсегда (код 410).

  Директива `RedirectMatch` позволяет использовать регулярные выражения.

  Пример:

```
Redirect permanent /file1.html /file2.html
RedirectMatch 301 ^/manual(?:/(de|en|es|ru)){2,}(/.*)?$
/manual/$1$2
```

- `RedirectPermanent` — выполняет перенаправление с состоянием `permanent`:

```
RedirectPermanent /file1.html /file2.html
```

- `RedirectTemp` — выполняет перенаправление с состоянием `temp`:

```
RedirectTemp /file1.html /temp_file.html
```

4.4.9. Обработка ошибок

С помощью директивы `ErrorDocument` можно указать документ, который будет выдан Web-браузеру при возникновении указанной ошибки:

```
ErrorDocument 404 /err/error404.html
```

Обычно указываются директивы (и разрабатываются соответствующие документы) для следующих ошибок:

- 401 — пользователь не авторизован;
- 403 — нет доступа. При отсутствии индексного файла в каталоге и отключенной опции `Indexes` директивы `Options` генерируется именно эта ошибка;
- 404 — ресурс не найден.

4.4.10. Настройки MIME-типов

При передаче файла сервер указывает MIME-тип документа. Это позволяет Web-браузеру правильно обработать получаемый файл. MIME-тип указывается в формате:

```
<Категория>/<Тип файла>
```

Примеры:

- `text/html` — для HTML-документов;
- `image/gif` — для изображений в формате GIF;
- `application/msword` — для документов в формате Word;
- `audio/mpeg` — для аудиофайлов MP3.

Конфигурации MIME-типов находятся в файле mime.types (C:\xampp\apache\conf\mime.types). Для настройки MIME-типов и смежных вопросов используются следующие директивы:

- `AddEncoding` — устанавливает взаимосвязь между определенной кодировкой и расширением файла:

  ```
  AddEncoding pkzip .zip
  ```

- `RemoveEncoding` — удаляет взаимосвязь между определенной кодировкой и расширением файла:

  ```
  RemoveEncoding .zip
  ```

- `TypesConfig` — указывает расположение конфигурационного файла с настройками MIME-типов:

  ```
  TypesConfig conf/mime.types
  ```

- `AddType` — позволяет добавить новый MIME-тип и связать его с определенным расширением:

  ```
  AddType text/html .shtml
  ```

- `RemoveType` — удаляет связи между MIME-типами и расширениями:

  ```
  RemoveType .cgi
  ```

- `ForceType` — указывает MIME-тип для набора файлов. Присваивает файлам, указанным в разделе `<Directory>` или `<Files>`, определенный MIME-тип, не принимая во внимание расширения файлов;
- `AddHandler` — связывает определенный обработчик с файловым расширением:

  ```
  AddHandler type-map .var
  ```

- SetHandler — обеспечивает обработку файлов в разделах <Directory> или <Files> с помощью определенного обработчика:

```
<Files *.html>
   SetHandler type-map
</Files>
```

- RemoveHandler — отменяет связывание определенного обработчика с файловым расширением:

```
RemoveHandler .var
```

В директивах AddHandler и SetHandler могут быть указаны следующие обработчики:

- default-handler — обработчик по умолчанию, который используется для обслуживания HTML-документов, файлов изображений (т. е. файлов, не требующих предварительной обработки);
- send-as-is — посылает файл, содержащий в себе HTTP-заголовки, как есть (без добавления пакетных или HTTP-заголовков). Заголовки можно указывать в самом файле, отделяя их от основного содержимого пустой строкой;
- cgi-script — обрабатывает файл как CGI-скрипт;
- imap-file — обрабатывает файл как карту-изображение;
- server-info — возвращает конфигурационную информацию сервера. Необходимо, чтобы был подключен модуль mod_info.so:

```
<Location /info>
   SetHandler server-info
</Location>
```

- server-status — возвращает отчет о состоянии сервера. Необходимо, чтобы был подключен модуль mod_status.so:

```
<Location /status>
   SetHandler server-status
</Location>
```

- type-map — обрабатывает файл как файл сопоставления типов:

```
AddHandler type-map .var
```

В этом примере все файлы с расширением var будут использоваться как файлы сопоставления типов.

Пример файла сопоставления типов:

```
URI: index.html.en
Content-Language: en
Content-type: text/html; charset=ISO-8859-1
URI: index.html.ru.koi8-r
Content-Language: ru
Content-type: text/html; charset=KOI8-R
```

- `Action` — устанавливает соответствие между заданным названием обработчика или MIME-типа с определенной программой, обеспечивающей механизм исполнения. Эта директива позволяет создавать собственные обработчики:

```
Action image/gif /cgi-bin/images.cgi

Action my-file-type /cgi-bin/program.cgi
AddHandler my-file-type .xyz
```

- `CacheNegotiatedDocs` — задает режим кэширования сервером результатов переговоров. Если директива имеет значение `on`, то документы, установленные в результате переговоров между сервером и Web-браузером о согласовании MIME-типа, языка и способа кодирования, могут быть помещены в кэш:

```
CacheNegotiatedDocs on
```

По умолчанию директива имеет значение `off`.

4.4.11. Управление листингом каталога

Управлять отображением листинга каталога позволяют следующие директивы:

- `DirectoryIndex` — задает название документа, который будет возвращен по запросу, если не указано название документа (например, **http://localhost/**):

```
DirectoryIndex index.php index.html
```

Если вначале указан файл index.php, а затем index.html, то при наличии двух файлов будет возвращен файл index.php. Чтобы в этом случае получить содержимое файла index.html, нужно его название указать явным образом: `http://localhost/index.html`.

Значение `disabled` отключает эту функцию;

- `IndexOptions` — определяет способ генерирования листинга каталога с помощью опций. Если опция помечена знаком `+`, то она добавляется к числу уже включенных опций, а если знаком `-`, то она отключается. Для использования этой директивы необходимо, чтобы опция `Indexes` директивы `Options` была включена. Могут быть указаны следующие опции:

  - `DescriptionWidth` — задает ширину столбца описания в символах. Если указан знак `*`, то ширина столбца станет равной ширине самого длинного описания:

```
IndexOptions +DescriptionWidth=30
IndexOptions +DescriptionWidth=*
```

  - `FancyIndexing` — включает режим, в котором листинг каталога будет иметь интерфейс, напоминающий диспетчер файлов;
  - `FoldersFirst` — устанавливает, что вначале отображаются названия каталогов, а затем названия файлов;

- HTMLTable — заставляет оформлять листинг каталога как HTML-таблицу в заданном формате, а не как список;
- IconsAreLinks — инструктирует сделать пиктограммы ссылками;
- IconWidth и IconHeight — задают размеры пиктограмм, отображаемых в листинге каталога (по умолчанию 20×22 пикселов):

```
IndexOptions +IconWidth=20 +IconHeight=22
```

- IgnoreCase — позволяет игнорировать регистр символов;
- IgnoreClient — отключает пересортировку листинга файлов по столбцам;
- NameWidth — устанавливает максимальную длину имени файла, отображаемую в листинге. Если указан знак *, то используется длина самого длинного имени файла;
- ScanHTMLTitles — предписывает отображать в описании файла информацию из тега <title>;
- SuppressColumnSorting — отключает сортировку листинга файлов по столбцам;
- SuppressDescription — удаляет столбец с описанием файлов;
- SuppressHTMLPreamble — удаляет стандартные открывающие и закрывающие теги (<html> и <body>). Применяется, если заданы директивы HeaderName и ReadmeName. Указанные этими директивами файлы должны иметь открывающие теги (для файла, указанного в HeaderName) и закрывающие (для файла, указанного в ReadmeName);
- SuppressIcon — выключает отображение пиктограмм в листинге каталога;
- SuppressLastModified — удаляет столбец с датой и временем последнего обновления файла;
- SuppressRules — отключает вывод разделительных линий сверху и снизу листинга;
- SuppressSize — удаляет столбец с размерами файлов;
- TrackModified — включает кэширование листинга каталога;
- VersionSort — устанавливает режим сортировки файлов с учетом номера версии;
- XHTML — задает формат XHTML 1.0 Transitional (по умолчанию используется HTML 3.2);
- Charset — задает кодировку;

❒ AddIcon — задает пиктограмму для названия файла или его части (например, расширения):

```
AddIcon /icons/binary.gif .bin .exe
```

- `AddIconByType` — задает пиктограмму для MIME-типов:

```
AddIconByType (TXT,/icons/text.gif) text/*
```

- `DefaultIcon` — устанавливает пиктограмму, используемую по умолчанию:

```
DefaultIcon /icons/unknown.gif
```

- `AddIconByEncoding` — связывает пиктограмму с типом кодировки:

```
AddIconByEncoding (CMP,/icons/compressed.gif) x-compress x-gzip
```

- `AddDescription` — устанавливает описание для файла или набора файлов, соответствующих шаблону:

```
AddDescription "Описание файла" name.html
```

Описание отображается в столбце **Описание листинга каталога**. Оно может включать HTML-форматирование;

- `HeaderName` — позволяет изменить стандартный заголовок листинга каталога:

```
HeaderName HEADER.html
```

Если указана опция `SuppressHTMLPreamble`, то содержимое файла заменит весь верхний колонтитул;

- `ReadmeName` — позволяет изменить стандартный нижний колонтитул листинга каталога:

```
ReadmeName README.html
```

Если указана опция `SuppressHTMLPreamble`, то содержимое файла заменит весь нижний колонтитул;

- `IndexIgnore` — служит для указания файлов, которые не должны быть показаны в листинге каталога:

```
IndexIgnore HEADER* README* .htaccess
```

- `IndexOrderDefault` — позволяет изменить первоначальную сортировку листинга каталога (по умолчанию файлы сортируются по имени). Первый аргумент задает порядок сортировки. Может принимать два значения: `Ascending` (по возрастанию) и `Descending` (по убыванию). Второй аргумент задает имя поля: `Name`, `Date`, `Size` или `Description`:

```
IndexOrderDefault Descending Date
```

- `IndexStyleSheet` — указывает таблицу стилей, задающую оформление для сгенерированной сервером страницы списка файлов:

```
IndexStyleSheet "/css/list.css"
```

- `IndexHeadInsert` — задает дополнительный HTML-код, который будет вставлен в тег `<head>` сгенерированной страницы со списком файлов:

```
IndexHeadInsert "<link rel=\"shortcut icon\"
href=\"/icons/favicon.ico\">"
```

4.4.12. Директивы протоколирования

Как уже упоминалось, события, происходящие на сервере, регистрируются Apache в журналах. По умолчанию в каталоге logs (C:\xampp\apache\logs) расположены два файла журналов: access.log и error.log. Эти журналы позволяют получить подробную информацию обо всех запросах и ошибках. Открыть эти файлы можно с помощью любого текстового редактора (например, Notepad++).

Файл access.log содержит следующую информацию: IP-адрес, дату и время запроса, метод (`GET` или `POST`), имя запрошенного файла, протокол, код состояния запроса (код 200 означает, что файл успешно найден, а 404 — что файл не найден) и размер файла. Кроме того, файл может содержать информацию о ссылающейся странице (с которой перешел пользователь на наш сайт с другого сайта), а также информацию о Web-браузере посетителя.

Вот пример строки журнала при успешной обработке запроса:

```
::1 - - [17/Sep/2017:00:31:02 +0300] "GET /index.php HTTP/1.1" 200 10 "-"
"Mozilla/5.0 (Windows NT 6.2; WOW64; rv:32.0)"
```

А вот пример строки журнала при отсутствии файла (ошибка 404):

```
::1 - - [17/Sep/2017:00:31:10 +0300] "GET /m HTTP/1.1" 404 109 "-"
"Mozilla/5.0 (Windows NT 6.2; WOW64; rv:32.0)"
```

Файл error.log содержит информацию об ошибках сервера Apache, а также различные предупреждения. Вот пример строки:

```
[Sat Sep 16 23:57:01.276330 2017] [ssl:warn] [pid 9156:tid 264] AH01909:
www.example.com:443:0 server certificate does NOT include an ID which
matches the server name
```

Местоположение и формат журналов задаются с помощью следующих директив:

- `CustomLog` — указывает, где расположен журнал регистрации, а также его формат:

  ```
  CustomLog "logs/access.log" common
  ```

- `LogFormat` — определяет фактический формат журнала регистрации. Псевдоним формата (`common`) указывается в директиве `CustomLog`:

  ```
  LogFormat "%h %l %u %t \"%r\" %>s %b" common
  ```

  В строке формата могут присутствовать следующие символы, которые заменяются фактическими значениями:

  - `%h` — адрес удаленного хоста (адрес клиента, сделавшего запрос);
  - `%l` — удаленное имя пользователя. Практически всегда содержит прочерк;
  - `%u` — имя пользователя, прошедшего аутентификацию;
  - `%t` — дата и время запроса;
  - `%r` — метод, имя запрошенного ресурса и протокол;

- %>s — статус запроса;
- %b — число отправленных байтов;
- %{Referer}i — страница, с которой пришел клиент;
- %{User-Agent}i — Web-браузер, используемый клиентом.

Существуют и другие переменные директивы LogFormat, но они встречаются крайне редко, т. к. программы обработки log-файлов настроены на форматы common и combined. С помощью этих программ можно получить статистические данные в более удобном формате;

- ErrorLog — определяет местоположение журнала регистрации ошибок:

```
ErrorLog "logs/error.log"
```

- LogLevel — позволяет установить уровень регистрации ошибок и диагностических сообщений в журнале error.log. По умолчанию директива настроена на регистрацию аварийных ситуаций (warn). Могут быть заданы следующие значения: debug, info, notice, warn, error, crit, alert или emerg:

```
LogLevel warn
```

- HostnameLookups — если директива имеет значение On, то Apache будет регистрировать полное имя хоста клиента, а не только IP-адрес. Значение по умолчанию:

```
HostnameLookups Off
```

4.4.13. Файл конфигурации .htaccess. Управляем сервером Apache из обычного каталога

На виртуальном хостинге не предоставляется доступ к главному файлу конфигурации, т. к. один сервер может обслуживать множество сайтов, принадлежащих различным людям. В этом случае для конфигурирования отдельных каталогов используется файл .htaccess. При изменении этого файла нет необходимости перезагружать сервер — файлы .htaccess анализируются при каждом запросе файла из каталога.

Если сервер находится в полном нашем распоряжении, то настраивать конфигурацию необходимо в файле httpd.conf, а использование файлов .htaccess нужно запретить, поскольку это сильно влияет на производительность и защиту. Дело в том, что, как уже отмечалось, файл httpd.conf анализируется только один раз (при запуске сервера), а файлы .htaccess — при каждом запросе. Если использование файлов .htaccess запрещено, то Apache даже не будет искать эти файлы в каталогах.

Для настройки файлов .htaccess предназначены следующие директивы:

- AccessFileName — задает имя файла конфигурации:

```
AccessFileName .htaccess
```

- AllowOverride — позволяет ограничить перечень директив, которые позволено изменять в файлах .htaccess. Директива может принимать следующие значения:

- `All` — позволяет пользователям переопределять в файлах .htaccess глобальные параметры доступа:

  ```
  AllowOverride All
  ```

- `None` — отключает использование файла .htaccess:

  ```
  AllowOverride None
  ```

- `AuthConfig` — разрешает директивы авторизации (`AuthName`, `AuthType`, `AuthUserFile`, `AuthGroupFile`, `Require` и др.):

  ```
  AllowOverride AuthConfig
  ```

- `FileInfo` — разрешает директивы, управляющие типами документов (`AddType`, `AddLanguage`, `AddEncoding`, `ErrorDocument`, `LanguagePriority` и др.):

  ```
  AllowOverride FileInfo
  ```

- `Indexes` — позволяет использование директив, управляющих индексацией каталога (`AddIcon`, `DirectoryIndex`, `FancyIndexing`, `HeaderName` и др.):

  ```
  AllowOverride Indexes
  ```

- `Limit` — делает возможным использование директив, управляющих доступом к хостам (`Allow`, `Deny` и `Order`):

  ```
  AllowOverride FileInfo AuthConfig Limit
  ```

- `Options` — разрешает директивы, управляющие каталогами (`Options` и `XbitHack`):

  ```
  AllowOverride Options
  ```

4.4.14. Защита содержимого папки паролем

Ограничить доступ к определенной папке можно с помощью следующих директив:

- `AuthType` — задает тип аутентификации. Параметр `Basic` указывает на базовую аутентификацию по имени пользователя и паролю:

  ```
  AuthType Basic
  ```

- `AuthName` — определяет текст, который будет отображен во всплывающем окне запроса:

  ```
  AuthName "Restricted area"
  ```

- `AuthUserFile` — указывает местоположение файла паролей:

  ```
  AuthUserFile "${SRVROOT}/data/pass.conf"
  ```

- `AuthGroupFile` — определяет местоположение файла групп;
- `Require` — задает дополнительные требования, которые должны быть выполнены для предоставления доступа. Могут быть указаны следующие параметры:
  - `valid-user` — доступ предоставляется любому пользователю, имя которого задано в файле, указанном директивой `AuthUserFile`, при условии правильно введенного пароля;

- `user` — доступ разрешается только указанным пользователям;
- `group` — доступ разрешается только указанным группам пользователей.

Ограничить доступ к определенной папке можно двумя способами:

- отредактировав файл конфигурации сервера (httpd.conf) — путь к защищаемой папке необходимо указать в разделе `<Directory>`:

```
<Directory "C:/xampp/htdocs/test">
   AuthType Basic
   AuthName "Restricted area"
   AuthUserFile "${SRVROOT}/data/pass.conf"
   <Limit GET POST>
      Require valid-user
   </Limit>
</Directory>
```

- разместив в защищаемой папке файл .htaccess с такими директивами:

```
AuthType Basic
AuthName "Restricted area"
AuthUserFile "C:/xampp/apache/data/pass.conf"
<Limit GET POST>
   Require valid-user
</Limit>
```

На виртуальном хостинге доступен только второй способ — предполагающий использование файла .htaccess. На своем локальном компьютере должна быть включена поддержка этого файла в главном файле конфигурации. Чтобы это проверить, находим раздел:

```
<Directory "C:/xampp/htdocs">
...
</Directory>
```

Внутри этого раздела директива `AllowOverride` должна иметь значение `All` (значение по умолчанию в пакете XAMPP):

```
AllowOverride All
```

Если значение было изменено, то сохраняем файл и перезапускаем сервер Apache, чтобы изменения вступили в силу. Затем открываем Notepad++ и набираем приведенный ранее код. Сохраняем набранный текст под названием .htaccess, предварительно создав каталог (например, test) в каталоге C:\xampp\htdocs. Создаем любой HTML-документ и сохраняем его в каталоге test под именем index.html в кодировке UTF-8. Содержимое этого файла будет отображаться при успешном входе в каталог.

Теперь сформируем файл паролей. Для этого создадим каталог data в каталоге C:\xampp\apache. Обратите внимание: мы будем сохранять файл вне корневого каталога документов сервера — файл паролей не должен быть доступен через Web-интерфейс.

Создать файл паролей (pass.conf) можно с помощью программы htpasswd.exe, расположенной в папке bin (C:\xampp\apache\bin). Для выполнения программы необходима командная строка. Запускаем командную строку и переходим в каталог C:\xampp\apache\bin:

```
C:\Users\Unicross>cd C:\xampp\apache\bin
```

В командной строке мы увидим приглашение:

```
C:\xampp\apache\bin>
```

В строке приглашения набираем команду, которая создаст файл C:\xampp\apache\data\pass.conf и добавит в него информацию о пользователе `user1`:

```
htpasswd -c C:\xampp\apache\data\pass.conf user1
```

Нажимаем клавишу <Enter> — появится приглашение ввести пароль:

```
C:\xampp\apache\bin>htpasswd -c C:\xampp\apache\data\pass.conf user1
New password:
```

Вводим пароль (например, `pass1`) и нажимаем клавишу <Enter> — программа попросит повторить пароль:

```
C:\xampp\apache\bin>htpasswd -c C:\xampp\apache\data\pass.conf user1
New password: *****
Re-type new password:
```

Повторяем и нажимаем клавишу <Enter>:

```
C:\xampp\apache\bin>htpasswd -c C:\xampp\apache\data\pass.conf user1
New password: *****
Re-type new password: *****
Adding password for user user1
```

В итоге будет в каталоге data создан файл pass.conf со следующими данными:

```
user1:$apr1$J7s/aTmT$gxaZKa8t1wGBJMRe6MTLP0
```

Как видим, пароль `pass1` в этом файле отсутствует — точнее, он присутствует в зашифрованном виде. Тем не менее, чтобы увеличить безопасность сервера, файлы с паролями следует сохранять в каталогах, не доступных извне, как мы и сделали.

Попробуем теперь создать пароль еще для одного пользователя. Для этого в командной строке набираем:

```
htpasswd -b C:\xampp\apache\data\pass.conf user2 pass2
```

Обратите внимание: флаг `-c` мы заменили флагом `-b`, а также указали пароль сразу после имени пользователя. Если использовать флаг `-c`, то файл будет перезаписан, и, соответственно, вся старая информация удалена.

После нажатия клавиши <Enter> информация о новом пользователе и его пароле будет добавлена в конец файла pass.conf, который станет выглядеть примерно так:

```
user1:$apr1$J7s/aTmT$gxaZKa8t1wGBJMRe6MTLP0
user2:$apr1$ryD1631F$z/j0DWeXgxde9BztH.Ai11
```

```
Командная строка
Microsoft Windows [Version 6.2.9200]
(c) Корпорация Майкрософт, 2012. Все права защищены.

C:\Users\Unicross>cd C:\xampp\apache\bin

C:\xampp\apache\bin>htpasswd -c C:\xampp\apache\data\pass.conf user1
New password: *****
Re-type new password: *****
Adding password for user user1

C:\xampp\apache\bin>htpasswd -b C:\xampp\apache\data\pass.conf user2 pass2
Adding password for user user2

C:\xampp\apache\bin>_
```

Рис. 4.13. Создание паролей в командной строке

Последовательность ввода инструкций в командной строке показана на рис. 4.13.

Открываем Web-браузер и в адресной строке набираем: `http://localhost/test/`. Если все сделано правильно, то при попытке открыть любой документ в этой папке будет выведено окно для ввода пароля (рис. 4.14).

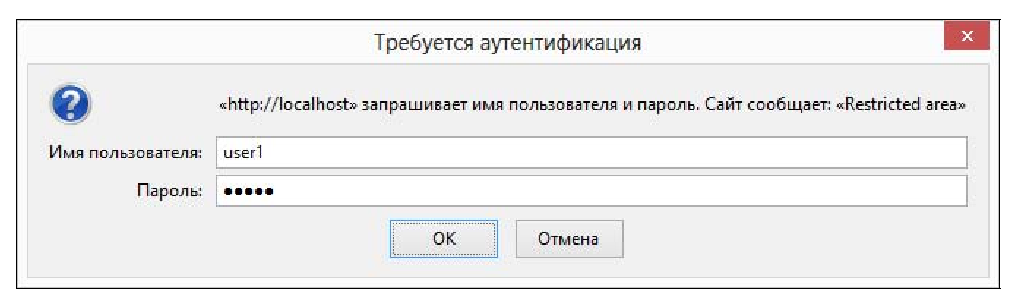


Рис. 4.14. Окно для ввода пароля

> **Примечание**
>
> Не рекомендуется набирать пароли в командной строке, поскольку введенные таким образом команды сохраняются в истории командной строки в незашифрованном виде и могут стать доступными злоумышленникам. Поэтому нужно не лениться и набирать пароли в ответ на приглашение программы *htpasswd.exe*.

4.4.15. Управление доступом

Директива `Require` пригодится нам и в случае, если мы хотим управлять доступом к папке в зависимости от IP-адреса, с которого поступил запрос, или метода. Здесь мы будем использовать один из следующих параметров:

- `all granted` — доступ разрешен для всех:

  ```
  Require all granted
  ```

- `all denied` — доступ запрещен для всех:

  ```
  Require all denied
  ```

- `ip` — разрешен доступ лишь с указанных IP-адресов или подсетей:

  ```
  Require ip 192.168 10.0.0.2
  ```

  Разрешает доступ лишь с IP-адресов из подсети 192.168.* и адреса 10.0.0.2;

- `host` — разрешен доступ лишь с указанных доменных имен:

  ```
  Require host www.test.ru
  ```

- `method` — разрешен доступ лишь с применением указанных методов:

  ```
  Require method POST
  ```

Вставив между наименованием директивы `Require` и остальными ее параметрами слово `not`, мы укажем, что при выполнении указанного нами условия доступ должен, напротив, не предоставляться. Обратите внимание: слово `not` можно указывать только внутри раздела `<RequireAll>`.

Предоставим доступ со всех IP-адресов за исключением находящихся в подсети 192.168.*:

```
<RequireAll>
   Require all granted
   Require not ip 192.168
</RequireAll>
```

Если нам нужно указать сложное условие, состоящее из нескольких простых условий, то следует воспользоваться следующими разделами:

- `RequireAll` — должны выполняться условия, заданные во всех директивах `Require`:

  ```
  AuthType Basic
  AuthName "Restricted area"
  AuthUserFile "${SRVROOT}/data/pass.conf"
  <RequireAll>
     Require ip 192.168
     Require user user1
  </RequireAll>
  ```

  Здесь мы предоставляем доступ к папке лишь пользователю `user1` при условии, что он обратился с IP-адреса, находящегося в подсети 192.168.*;

- `RequireAny` — должно выполняться хотя бы одно условие из заданных в директивах `Require`:

  ```
  AuthType Basic
  AuthName "Restricted area"
  ```

```
AuthUserFile "${SRVROOT}/data/pass.conf"
<RequireAny>
   Require ip 192.168
   Require user user1
</RequireAny>
```

Предоставляем доступ к папке пользователю `user1` и всем прочим пользователям, обратившимся с IP-адреса, который находится в подсети 192.168.*;

- `RequireNone` — не должно выполняться ни одно из условий, заданных в директивах `Require`. Раздел `<RequireNone>` может находиться только внутри другого раздела — например, внутри раздела `<RequireAll>`:

```
AuthType Basic
AuthName "Restricted area"
AuthUserFile "${SRVROOT}/data/pass.conf"
<RequireAll>
   Require valid-user
   <RequireNone>
      Require user user2
      Require user user3
   </RequireNone>
</RequireAll>
```

Предоставляем доступ к папке всем авторизованным пользователям, за исключением `user2` и `user3`.

> ***Внимание!***
>
> Директива `Require` и разделы `<RequireAll>`, `<RequireAny>` и `<RequireNone>` в конфигурационном файле *httpd.conf* должны быть расположены внутри раздела `<Directory>`. В файле *.htaccess* раздел `<Directory>` указывать не нужно.

> ***Примечание***
>
> В предыдущих версиях Apache для управления доступом использовались директивы `Order`, `Allow` и `Deny`. Эти директивы доступны и в версии 2.4, но они объявлены устаревшими и не рекомендуются к использованию.

4.4.16. Регулярные выражения, используемые в директивах

Некоторые директивы позволяют использовать регулярные выражения. Эти выражения мало чем отличаются от регулярных выражений JavaScript (см. *разд. 3.8*). В них допустимы следующие метасимволы и специальные конструкции:

- `^` — привязка к началу строки;
- `$` — привязка к концу строки;
- `[]` — позволяет указать символы, которые могут встречаться на этом месте в строке. Можно привести символы подряд или указать диапазон через тире;

- [^] — значение можно инвертировать, если после первой скобки указать символ ^. Так можно указать символы, которых не должно быть на этом месте в строке.

  Для использования специальных символов в качестве обычных необходимо перед специальным символом указать символ \;
- \d — любая цифра;
- \w — любая латинская буква, цифра или знак подчеркивания;
- \s — любой непечатаемый символ (пробел, табуляция, перевод страницы, новая строка или перевод каретки);
- . (точка) — любой символ, кроме символа перевода строки (\n);
- \D — не цифра;
- \W — не латинская буква, не цифра и не знак подчеркивания;
- \S — не непечатаемый символ;
- {n} — в точности n вхождений предыдущего символа или подвыражения в строку;
- {n,} — n или более вхождений символа в строку;
- {n,m} — не менее n вхождений символа в строку и не более m. Цифры указываются через запятую без пробела;
- * — ноль или большее число вхождений символа в строку;
- + — один или большее число вхождений символа в строку;
- ? — ноль или одно число вхождений символа в строку;
- n|m — один из символов n или m.

Регулярное выражение можно разбить на подвыражения с помощью круглых скобок. Каждая группа символов, соответствующих подвыражению, сохраняется в памяти. В дальнейшем группу символов можно извлечь, указав после символа $ номер скобки:

```
AliasMatch ^/manual(?:/(?:de|en|ru))?(/.*)?$ "C:/xampp/apache/manual$1"
```

4.4.17. Создание виртуальных серверов

Виртуальные серверы создаются с помощью раздела <VirtualHost> и позволяют размещать на одном сервере несколько сайтов.

Попробуем создать на сервере два новых сайта: один сайт будет доступен по IP-адресу 127.0.0.2 и имени site1, а второй — по IP-адресу 127.0.0.3 и имени site2. Для этого в каталоге C:\xampp\apache создаем два каталога: site1 и site2.

В каталог site1 добавляем файл index.html следующего, например, содержания:

```
<!DOCTYPE html>
<html lang="ru">
<head>
```

```
   <meta charset="utf-8">
   <title>Новый сайт 1</title>
</head>
<body>Это сайт 1</body>
</html>
```

В каталог site2 добавляем файл index.html следующего содержания:

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Новый сайт 2</title>
</head>
<body>Это сайт 2</body>
</html>
```

Открываем файл httpd-vhosts.conf (который расположен в папке C:\xampp\apache\conf\extra). Все инструкции в этом файле должны быть закомментированы. Добавим в конец файла следующие строки:

```
<VirtualHost *:80>
   DocumentRoot "C:/xampp/htdocs"
   ServerName localhost
</VirtualHost>

<VirtualHost 127.0.0.2:80>
   ServerAdmin webmaster@site1
   DocumentRoot "${SRVROOT}/site1"
   ServerName site1
</VirtualHost>
<Directory "${SRVROOT}/site1">
   Options -Indexes +Includes +FollowSymLinks
   AllowOverride All
   Require all granted
   AddDefaultCharset UTF-8
</Directory>

<VirtualHost 127.0.0.3:80>
   ServerAdmin webmaster@site2
   DocumentRoot "${SRVROOT}/site2"
   ServerName site2
</VirtualHost>
<Directory "${SRVROOT}/site2">
   Options -Indexes +Includes +FollowSymLinks
   AllowOverride All
   Require all granted
   AddDefaultCharset UTF-8
</Directory>
```

Сохраняем и закрываем файл. Теперь его необходимо подключить к главному конфигурационному файлу httpd.conf. Открываем файл httpd.conf и проверяем, убран ли символ комментария (`#`) перед строкой:

```
Include conf/extra/httpd-vhosts.conf
```

Сохраняем файл httpd.conf и перезагружаем сервер. Теперь открываем Web-браузер и в адресной строке набираем: `http://127.0.0.3/`. В итоге в окне Web-браузера должна отобразиться надпись **Это сайт 2**.

Для того чтобы можно было использовать доменные имена (`site1` и `site2`), необходимо в конец файла hosts (расположенного в папке C:\Windows\System32\Drivers\etc) дописать две строки:

```
127.0.0.2       site1
127.0.0.3       site2
```

> ***Внимание!***
>
> Для редактирования файла *hosts* необходимы права администратора. Через поиск находим приложение Блокнот, щелкаем на нем правой кнопкой мыши и выбираем пункт **Запуск от имени администратора**. В меню **Файл** выбираем пункт **Открыть** и находим файл. Открываем его, вставляем строки и сохраняем файл.

Теперь открываем Web-браузер и в адресной строке набираем: `http://site2/`. В итоге в окне Web-браузера снова должна появиться надпись **Это сайт 2**.

Теперь нам доступны три виртуальных хоста: `localhost`, `site1` и `site2`. Причем они расположены на разных IP-адресах и не зависят друг от друга. По аналогии можно создать и другие хосты.

> ***Внимание!***
>
> Название виртуального хоста необходимо указывать без точки. Например, `site1`, а не `site1.ru`. В противном случае вы не сможете попасть на реальный сайт **site1.ru**, не удалив строку из файла *hosts*.

4.5. Файл конфигурации PHP (php.ini)

Файл php.ini (C:\xampp\php\php.ini) — это основной файл конфигурации PHP. Открыть и отредактировать его можно с помощью любого текстового редактора, например Блокнота или Notepad++.

Если в начале строки указан символ `;`, то такая строка является комментарием:

```
; See the PHP docs for more specific information.
; http://php.net/manual/ru/ini.php
```

Давайте внесем изменения в файл конфигурации. Вначале проверьте значения следующих директив:

```
extension_dir="C:/xampp/php/ext"
default_charset="UTF-8"
```

```
session.save_path="C:/xampp/tmp"
display_errors=On
short_open_tag=Off
upload_tmp_dir="C:/xampp/tmp"
```

Далее необходимо проверить подключение библиотек (перед директивами не должно быть символа комментария):

```
extension=mysqli
extension=gd2
extension=curl
extension=mbstring
```

> ***Примечание***
> В предыдущих версиях PHP строки имеют следующий формат:
> `extension=php_mysqli.dll`.

Находим строку:

```
include_path=C:\xampp\php\PEAR
```

и меняем ее на:

```
include_path=".;C:/xampp/php/includes;C:/xampp/php/PEAR"
```

Создаем в каталоге C:\xampp\php каталог includes — в нем будут храниться подключаемые файлы.

Находим строку:

```
upload_max_filesize=2M
```

и увеличиваем максимально допустимый размер загружаемых файлов до 16 Мбайт:

```
upload_max_filesize=16M
```

Заменяем строку:

```
date.timezone=Europe/Berlin
```

на

```
date.timezone=Europe/Moscow
```

> ***Примечание***
> Выбрать название зоны для вашей местности можно на странице **http://php.net/manual/ru/timezones.php**.

Включаем вывод всех сообщений об ошибках. Для этого меняем строку:

```
error_reporting=E_ALL & ~E_DEPRECATED & ~E_STRICT
```

на

```
error_reporting=E_ALL
```

Проверяем наличие папки C:\xampp\php\logs. Если папки logs нет, то ее нужно создать. В этой папке будет создан файл php_error_log, в который будут записываться

все сообщения об ошибках в PHP-скриптах. Путь и название файла прописываются в директиве `error_log`:

```
error_log="C:/xampp/php/logs/php_error_log"
```

Проверяем значения следующих директив:

```
display_errors=On
log_errors=On
```

И в завершение сохраняем и закрываем файл php.ini. С учетом того, что мы будем использовать PHP как модуль сервера Apache, после редактирования файла php.ini следует перезагрузить сервер, чтобы все настройки были успешно применены.

Подключение PHP к серверу Apache производится в файле httpd-xampp.conf (он расположен в каталоге C:\xampp\apache\conf\extra). В файле можно найти следующие инструкции:

```
LoadFile "C:/xampp/php/php7ts.dll"
LoadFile "C:/xampp/php/libpq.dll"
LoadModule php7_module "C:/xampp/php/php7apache2_4.dll"

<FilesMatch "\.php$">
    SetHandler application/x-httpd-php
</FilesMatch>

<IfModule php7_module>
    PHPINIDir "C:/xampp/php"
</IfModule>
```

Чтобы проверить основные настройки, следует запустить код из листинга 4.2. Набираем код и сохраняем файл под именем index.php в каталоге C:\xampp\htdocs. Открываем Web-браузер и в адресной строке набираем: `http://localhost/`.

Листинг 4.2. Проверка корректности настройки PHP 7

```
<?php
$err = array();
if (!file_exists('C:\\xampp\\php\\php.ini'))
   $err[] = 'Файл C:\\xampp\\php\\php.ini не существует';
$path = php_ini_loaded_file();
if (strtolower($path) !== 'c:\\xampp\\php\\php.ini')
   $err[] = 'Пути к файлу php.ini не совпадают';
if (!file_exists('C:\\xampp\\php\\ext\\'))
   $err[] = 'Папка C:\\xampp\\php\\ext\\ не существует';
$ext = ini_get("extension_dir");
if (strtolower($ext) !== 'c:/xampp/php/ext')
   $err[] = 'Проверьте значение директивы extension_dir';
$inc = ini_get('include_path');
$inc_path = '.;c:/xampp/php/includes;c:/xampp/php/pear';
```

```
if (strtolower($inc) !== $inc_path)
   $err[] = 'Проверьте значение директивы include_path';
$ses = ini_get('session.save_path');
if (strtolower($ses) !== 'c:/xampp/tmp')
   $err[] = 'Проверьте значение директивы session.save_path';
if (!file_exists('C:\\xampp\\tmp\\'))
   $err[] = 'Папка C:\\xampp\\tmp\\ не существует';
if (!file_exists('C:\\xampp\\php\\includes\\'))
   $err[] = 'Папка C:\\xampp\\php\\includes\\ не существует';
$upl = ini_get('upload_tmp_dir');
if (strtolower($upl) !== 'c:/xampp/tmp')
   $err[] = 'Проверьте значение директивы upload_tmp_dir';
if (!extension_loaded('gd'))
   $err[] = 'Библиотека GD не подключена';
if (!extension_loaded('mbstring'))
   $err[] = 'Библиотека mbstring не подключена';
if (!extension_loaded('mysqli'))
   $err[] = 'Библиотека mysqli не подключена';
$reporting = ini_get('error_reporting');
if ($reporting !== "32767")
   $err[] = 'Проверьте значение директивы error_reporting';
if (count($err) == 0)
   echo 'Ошибок нет';
else {
   echo '<div style="color:red;">';
   echo implode('<br>', $err) . '</div>';
}
```

Если после выполнения кода было выведено сообщение **Ошибок нет** — значит, все настроено правильно, и можно приступать к изучению PHP.

4.6. Файл конфигурации MySQL (my.ini)

В состав пакета XAMPP входит сервер баз данных MySQL, а точнее его аналог, распространяемый под свободной лицензией, — MariaDB. У MySQL есть свой собственный конфигурационный файл с названием my.ini (он расположен в папке C:\xampp\mysql\bin). Открыть и отредактировать его можно с помощью любого текстового редактора, например Блокнота или Notepad++.

Если в начале строки указан символ #, то такая строка является комментарием. Вот пример инструкций из этого файла:

```
# The MySQL server
[mysqld]
port= 3306
socket = "C:/xampp/mysql/mysql.sock"
basedir = "C:/xampp/mysql"
```

```
tmpdir = "C:/xampp/tmp"
datadir = "C:/xampp/mysql/data"
pid_file = "mysql.pid"
```

Для работы с MySQL из программы, написанной на языке PHP, нужно, чтобы был запущен не только сервер Apache, но и сервер MySQL. Для этого открываем приложение XAMPP Control Panel (C:\xampp\xampp-control.exe) и нажимаем кнопки **Start** у пунктов **Apache** и **MySQL**. При успешном запуске названия серверов будут выделены зеленым цветом, а справа отобразятся идентификаторы процессов и номера портов, на которых были запущены серверы (см. рис. 4.8). При этом кнопки **Start** будут заменены на кнопки **Stop**, с помощью которых можно серверы остановить.

Теперь проверим MySQL на работоспособность. Для этого открываем Notepad++ и набираем код, приведенный в листинге 4.3.

Листинг 4.3. Вывод содержимого базы данных

```
<?php
if (@$db = mysqli_connect('localhost', 'root', '')) {
   mysqli_query($db, "SET NAMES `utf8`");
   $query = 'SHOW TABLES FROM `mysql`';
   if ($res = mysqli_query($db, $query)) {
      echo "<h2>Содержимое базы данных `mysql`</h2>\n";
      while ($row = mysqli_fetch_row($res)) {
         echo 'Таблица: ' . $row[0] . "<br>\n";
      }
   }
   else {
      echo 'Ошибка ' . mysqli_errno($db) . ' ' . mysqli_error($db);
   }
}
else {
   echo 'Не удалось подключиться к базе данных';
}
?>
```

Во второй строке кода с помощью функции `mysqli_connect()` производится подключение к серверу MySQL. В первом параметре функция принимает название хоста (`localhost`), во втором — имя пользователя (`root`), а в третьем — пароль пользователя. По умолчанию при установке пакета XAMPP для сервера MySQL создается пользователь с именем `root`, который не имеет пароля, поэтому в третьем параметре указана пустая строка. Если вы при настройке MySQL указали пароль для пользователя `root`, нужно соответствующим образом изменить вторую строку в набранном файле.

В третьей строке кода с помощью функции `mysqli_query()` задается кодировка соединения. В нашем примере указана кодировка UTF-8 (обратите внимание на отсутствие дефиса в имени кодировки в SQL-запросе). Если кодировку соединения не

указать явным образом, то русские буквы могут отображаться в виде знаков вопроса. Кодировкой UTF-8 мы будем пользоваться постоянно при работе с базами данных, поэтому не забудьте указать эту кодировку и для файла с программой (флажок **Кодировать в UTF-8 (без BOM)** в меню **Кодировки** в редакторе Notepad++).

Сохраняем файл под именем test.php в каталоге C:\xampp\htdocs. Открываем Web-браузер и в адресной строке набираем: `http://localhost/test.php`.

Если в окне Web-браузера отобразился текст, приведенный в листинге 4.4, значит, все в порядке.

Листинг 4.4. Содержимое базы данных

```
Содержимое базы данных `mysql`
Таблица: column_stats
Таблица: columns_priv
Таблица: db
Таблица: event
Таблица: func
Таблица: general_log
Таблица: gtid_slave_pos
Таблица: help_category
Таблица: help_keyword
Таблица: help_relation
Таблица: help_topic
Таблица: host
Таблица: index_stats
Таблица: innodb_index_stats
Таблица: innodb_table_stats
Таблица: ndb_binlog_index
Таблица: plugin
Таблица: proc
Таблица: procs_priv
Таблица: proxies_priv
Таблица: roles_mapping
Таблица: servers
Таблица: slave_master_info
Таблица: slave_relay_log_info
Таблица: slave_worker_info
Таблица: slow_log
Таблица: table_stats
Таблица: tables_priv
Таблица: time_zone
Таблица: time_zone_leap_second
Таблица: time_zone_name
Таблица: time_zone_transition
Таблица: time_zone_transition_type
Таблица: user
```

Если вместо этого текста отобразилось сообщение: `Не удалось подключиться к базе данных`, то проверьте, запущен ли сервер MySQL, а также параметры подключения во второй строке кода.

Если появилось сообщение типа:

```
Parse error: syntax error, unexpected 'if' (T_IF) in
C:\xampp\htdocs\test.php on line 5
```

то вы допустили какую-то ошибку при наборе кода — например, забыли поставить точку с запятой в конце строки или пропустили скобку.

Если отобразилось сообщение такого вида:

```
Ошибка 1049 Unknown database 'mysyql'
```

то неправильно указано имя базы данных.

Если отобразилось сообщение:

```
Fatal error: Uncaught Error: Call to undefined function mysqli_connect()
in C:\xampp\htdocs\test.php:2
```

значит, не подключена библиотека php_mysqli.dll. Убедитесь, что были исполнены все инструкции по настройке PHP, приведенные в *разд. 4.5*.

4.7. Программа phpMyAdmin

Программа phpMyAdmin позволяет наглядно работать с базами данных через Web-интерфейс. В пакете XAMPP программа установлена в папку C:\xampp\phpMyAdmin и доступна по адресу **http://localhost/phpmyadmin/**.

Для управления настройками программы предназначен конфигурационный файл config.inc.php, расположенный в каталоге C:\xampp\phpMyAdmin. В этом файле можно найти следующие строки:

```
$cfg['Servers'][$i]['auth_type'] = 'config';
$cfg['Servers'][$i]['user'] = 'root';
$cfg['Servers'][$i]['password'] = '';
$cfg['Servers'][$i]['extension'] = 'mysqli';
```

Первая строка задает тип идентификации. Значение `config` означает, что регистрационные данные пользователя будут браться из конфигурационного файла. Если эту строку закомментировать, то логин и пароль нужно будет вводить вручную. Во второй строке указывается имя пользователя, а в третьей — пароль этого пользователя. Четвертая строка задает имя библиотеки в PHP, используемой для выполнения запросов к базе данных.

Теперь попробуем поработать с программой. Открываем Web-браузер и в адресной строке набираем: `http://localhost/phpmyadmin/` — должно отобразиться окно, показанное на рис. 4.11.

Если все надписи выводятся по-английски, выберем в раскрывающемся списке **Language** пункт **Русский — Russian**.

В списке **Сопоставление кодировки соединения** задается кодовая таблица, используемая при работе с базами данных MySQL. Выбираем пункт **utf8mb4_unicode_ci**.

Попробуем создать новую базу данных. Для этого переключимся на вкладку **Базы данных** и в поле **Создать базу данных** наберем: `test2`. В списке **Сравнение** выбираем пункт **utf8_general_ci** и нажимаем кнопку **Создать**. Через некоторое время отобразится сообщение **База данных test2 была создана**. Вновь созданная база тотчас появится в иерархическом списке, расположенном слева, и в списке, что находится на вкладке **Базы данных**.

В левом иерархическом списке выбираем созданную базу `test2`. Переходим на вкладку **SQL**. В текстовом поле набираем следующий текст:

```
CREATE TABLE `city` (
  `id_city` int(11) NOT NULL auto_increment,
  `name_city` varchar(255) default NULL,
  PRIMARY KEY  (`id_city`)
) ENGINE=MyISAM;

INSERT INTO `city` (`id_city`, `name_city`) VALUES
(1, 'Санкт-Петербург'),
(2, 'Москва'),
(3, 'Новгород'),
(4, 'Тверь'),
(5, 'Минск');
```

Нажимаем кнопку **Вперед**. В итоге отобразится код заданного нами запроса со статистическими сведениями.

Теперь добавим нового пользователя для созданной базы данных. Для этого переходим по ссылке **Сервер: 127.0.0.1**. Далее выбираем вкладку **Учетные записи пользователей**. В открывшейся вкладке переходим по ссылке **Добавить учетную запись пользователя**. В поле **Имя пользователя** набираем: `petr`. В списке **Имя хоста** выбираем пункт **Локальный**. В поле **Пароль** набираем: `123`. Повторяем пароль в поле **Подтверждение**. Нажимаем кнопку **Вперед**.

Добавленный нами пользователь появится в списке, присутствующем на вкладке **Учетные записи пользователей**. Находим его и нажимаем расположенную в представляющем его пункте списка гиперссылку **Редактировать привилегии**. На открывшейся странице щелкаем гиперссылку **База данных**. В списке **Добавить привилегии на следующую базу данных** выбираем базу `test2` и нажимаем кнопку **Вперед** — отобразится окно **Редактирование привилегий**. Устанавливаем флажки во всех разделах (**Данные**, **Структура** и **Администрирование**) или сразу устанавливаем флажок **Отметить все**. Нажимаем кнопку **Вперед**.

После добавления пользователя необходимо перезагрузить привилегии. Для этого переходим по ссылке **Сервер: 127.0.0.1** и далее — по ссылке **Учетные записи пользователей**. Затем выбираем ссылку **Перезагрузить привилегии**. В итоге отобразится сообщение **Привилегии были успешно перезагружены**.

Попробуем отобразить все города из нашей базы данных в алфавитном порядке. Открываем Notepad++ и набираем код, приведенный в листинге 4.5.

Листинг 4.5. Вывод названий городов из базы данных

```
<?php
if (@$db = mysqli_connect('localhost', 'petr', '123', 'test2')) {
   mysqli_query($db, "SET NAMES `utf8`");
   $query = 'SELECT * FROM `city` ORDER BY `name_city`';
   if ($res = mysqli_query($db, $query)) {
      echo "Содержимое таблицы `city`<br><br>\n";
      while ($row = mysqli_fetch_assoc($res)) {
         echo $row['name_city'] . '<br>';
      }
   }
   else {
      echo 'Ошибка ' . mysqli_errno($db) . ' ' . mysqli_error($db);
   }
}
else {
   echo 'Не удалось подключиться к базе данных';
}
?>
```

Сохраняем файл под названием test2.php в каталоге C:\xampp\htdocs. Открываем Web-браузер и в адресной строке набираем: `http://localhost/test2.php`.

Если вы получили сообщение **Не удалось подключиться к базе данных**, то проверьте, запущен ли сервер MySQL, а также параметры подключения во второй строке кода. Попробуйте изменить вторую строку следующим образом:

```
if (@$db = mysqli_connect('localhost', 'root', '', 'test2')) {
```

Если после этого изменения вы получили список городов, то пользователь `petr` не был создан, или указан неправильный пароль, или привилегии для пользователя не заданы. Чтобы увидеть предупреждающие сообщения, уберите символ `@` перед именем переменной `$db`. Этот символ отключает вывод предупреждающих сообщений.

Результат выполнения кода из листинга 4.5 должен выглядеть следующим образом:

```
Содержимое таблицы `city`

Минск
Москва
Новгород
Санкт-Петербург
Тверь
```

Если в первой строке отображаются знаки вопроса вместо русских букв, то проверьте кодировку файла с программой, — она должна быть UTF-8. Если вместо названия городов отображаются знаки вопроса, то проверьте кодировку соединения с базой данных, — она также должна быть UTF-8. Кодировка соединения указывается в третьей строке:

```
mysqli_query($db, "SET NAMES `utf8`");
```

Для кодировки windows-1251 инструкция будет выглядеть следующим образом:

```
mysqli_query($db, "SET NAMES `cp1251`");
```

Если русские буквы все равно отображаются неправильно, то следует проверить кодировку, отправляемую сервером — она также должна быть UTF-8. Кодировку можно указать внутри программы явным образом. Для этого в самом начале программы, сразу после открывающего дескриптора `<?php`, нужно вставить перенос строки и следующую инструкцию:

```
header('Content-Type: text/html; charset=utf-8');
```

В предыдущих разделах мы настраивали кодировку, отправляемую сервером, поэтому, если после вставки этой строки, русские буквы стали отображаться правильно, то вы не выполнили инструкции по настройке.

Русские буквы должны отображаться правильно без всяких дополнительных инструкций. Если это так, то вы готовы изучать PHP и MySQL.



ГЛАВА 5

Основы PHP. Создаем динамические Web-страницы

5.1. Первые шаги

PHP — это язык программирования, выполняемый на стороне сервера. Программа на PHP, в отличие от программ на языке JavaScript, не зависит от программного обеспечения клиента и поэтому будет выполнена в любом случае.

Последовательность инструкций (называемая *программой* или *скриптом*) выполняется *интерпретатором* языка PHP. Код программы может внедряться в HTML-код. Эта возможность отличает PHP от других языков, используемых в Интернете, — например, от языка Perl. PHP-код обрабатывается на сервере до того, как страница будет передана Web-браузеру. В итоге Web-браузер получит обычный HTML-код или другой вывод.

> ***Примечание***
>
> В этом издании книги мы рассмотрим возможности PHP версии 7 без оглядки на версию 5 и более ранние версии. Если вы используете PHP 5, то для его изучения следует воспользоваться предыдущими изданиями книги.

5.1.1. Первая программа

При изучении языков программирования принято начинать с программы, выводящей надпись «Hello, world». Не будем нарушать традицию и продемонстрируем, как это выглядит на PHP (листинг 5.1).

Листинг 5.1. Первая программа

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
```

```
   <title>Первая программа</title>
</head>
<body>
<?php
echo "Hello, world";
?>
</body>
</html>
```

Набираем код в Notepad++ и сохраняем файл в формате PHP (под именем, например, index.php) в каталоге C:\xampp\htdocs. В меню **Кодировки** должен быть установлен флажок **Кодировать в UTF-8 (без BOM)**. Запускаем Web-браузер и в его адресной строке набираем: `http://localhost/`. В итоге в окне Web-браузера отобразится надпись **Hello, world**.

Теперь давайте рассмотрим исходный HTML-код (листинг 5.2).

Листинг 5.2. Исходный HTML-код

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Первая программа</title>
</head>
<body>
Hello, world</body>
</html>
```

Как нетрудно заметить, никаких признаков PHP в исходном коде нет.

HTML-теги также можно выводить с помощью оператора `echo`. Давайте заменим содержимое нашего файла на код листинга 5.3.

Листинг 5.3. Вывод HTML-тегов с помощью PHP

```
<?php
echo "<!DOCTYPE html>";
echo "<html lang=\"ru\">";
echo "<head>";
echo "   <meta charset=\"utf-8\">";
echo "   <title>Первая программа</title>";
echo "</head>";
echo "<body>";
echo "Hello, world";
echo "</body>";
echo "</html>";
?>
```

В итоге получим следующий исходный код:

```
<!DOCTYPE html><html lang="ru"><head>   <meta charset="utf-8">
<title>Первая программа</title></head><body>Hello, world</body></html>
```

Как можно видеть, в этом случае весь код отображается на одной строке. Чтобы отобразить каждый тег на отдельной строке, необходимо добавить символ перевода строки (листинг 5.4). Для системы UNIX таким символом будет `\n`. В операционной системе Windows символ перевода строки состоит из комбинации двух символов: `\r\n`.

Листинг 5.4. Вывод каждого тега на отдельной строке

```
<?php
echo "<!DOCTYPE html>\n";
echo "<html lang=\"ru\">\n";
echo "<head>\n";
echo "   <meta charset=\"utf-8\">\n";
echo "   <title>Первая программа</title>\n";
echo "</head>\n";
echo "<body>\n";
echo "Hello, world\n";
echo "</body>\n";
echo "</html>\n";
?>
```

Теперь каждый тег будет на своей строчке (листинг 5.5).

Листинг 5.5. Результат вывода предыдущей программы

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Первая программа</title>
</head>
<body>
Hello, world
</body>
</html>
```

При выводе HTML-тегов с помощью оператора `echo` следует помнить, что теги могут иметь параметры, значения которых заключаются в кавычки. Например, если попробовать вывести тег `<span>` так, как показано в листинге 5.6, то возникнет ошибка

```
Parse error: syntax error, unexpected 'color' (T_STRING), expecting ','
or ';' in C:\xampp\htdocs\index.php on line 8
```

Листинг 5.6. Ошибочный код при выводе кавычек

```
<?php
echo "<!DOCTYPE html>\n";
echo "<html lang=\"ru\">\n";
echo "<head>\n";
echo "   <meta charset=\"utf-8\">\n";
echo "   <title>Первая программа</title>\n";
echo "</head>\n<body>\n";
echo "<span style="color:red">";
echo "Hello, world";
echo "</span>\n";
echo "</body>\n</html>\n";
?>
```

Обойти эту проблему можно следующими способами:

- добавить защитный слеш перед каждой кавычкой:

  ```
  echo "<span style=\"color:red\">";
  ```

- в операторе `echo` использовать не кавычки, а апострофы:

  ```
  echo '<span style="color:red">';
  ```

> **Внимание!**
>
> Применение апострофов может повлечь за собой другие проблемы. Например, в этом случае нельзя использовать специальные символы (`\n`). Кроме того, если внутри присутствует переменная, то вместо ее значения мы увидим имя переменной.

Все инструкции в PHP заканчиваются точкой с запятой. В отличие от JavaScript, где отсутствие этого символа не приводит к сообщению об ошибке, отсутствие точки с запятой в PHP влечет за собой остановку выполнения сценария и выдачу сообщения об ошибке. Это самая распространенная ошибка среди начинающих изучать язык PHP.

Однако если после инструкции сразу идет закрывающий PHP-дескриптор, то точку с запятой можно не указывать:

```
<?php
echo "Hello, world"
?>
```

5.1.2. Особенности создания скриптов в кодировке UTF-8

В этом издании книги мы по умолчанию будем использовать кодировку UTF-8. Следует учитывать, что русские буквы в UTF-8 кодируются двумя байтами, а не одним, как, например, в кодировке windows-1251, а латинские буквы, цифры и специальные символы — одним байтом. При работе со строками в кодировке UTF-8

важно помнить, что функции, предназначенные для работы с однобайтовыми кодировками, могут хорошо работать с латинскими буквами, цифрами и специальными символами, но абсолютно неправильно с русскими буквами. Например, при обрезке строки могут появляться символы вопроса.

Сохранять все предлагаемые в книге файлы PHP-скриптов следует в кодировке UTF-8, если иное не оговорено отдельно. При этом нужно иметь в виду еще один момент. Ранее уже отмечалось, что при сохранении текстового файла в кодовой таблице UTF-8 Блокнот вставляет в его начало особые служебные символы, называемые сокращенно BOM (Byte Order Mark, метка порядка байтов). Эти символы не являются обязательными, и они не позволят нам установить заголовки ответа сервера с помощью функции `header()` (при условии отключения буферизации вывода). Поэтому сохранять файлы следует именно в программе Notepad++, установив перед набором кода в меню **Кодировки** флажок **Кодировать в UTF-8 (без BOM)**. При копировании кода через буфер обмена советуем вначале сохранить пустой файл в кодировке UTF-8 без BOM, вставить код из буфера обмена, а затем сохранить файл с помощью соответствующей кнопки на панели инструментов.

Если мы хотим создавать страницы в той кодовой таблице, которая указана в настройках сервера, нам не нужно выполнять никаких дополнительных действий. Но если планируется создание страниц в другой кодовой таблице, придется указать в программе эту кодовую таблицу явным образом:

- если сервер настроен на кодировку windows-1251, и планируется использование кодировки UTF-8, шаблон программы будет выглядеть так:

```
<?php
header('Content-Type: text/html; charset=utf-8');
// Сюда вставляем примеры в кодировке UTF-8
?>
```

- если же сервер настроен на кодировку UTF-8, но страницы будут создаваться в кодировке windows-1251, шаблон программы станет таким:

```
<?php
header('Content-Type: text/html; charset=windows-1251');
// Сюда вставляем примеры в кодировке windows-1251
?>
```

Напомним, что в *главе 4* мы выполнили настройку сервера на кодировку UTF-8. Поэтому при работе с этой кодировкой никаких дополнительных инструкций вставлять не нужно. Кроме того, в *разд. 4.3* мы создали в каталоге C:\xampp\htdocs два каталога: utf8 и cp1251. Каталог utf8 наследует все настройки кодировки от сервера. Внутри же каталога cp1251 сервер перенастроен на кодировку windows-1251. Поэтому, когда мы будем говорить, что файл с программой нужно сохранить в кодировке windows-1251, то нужно задать эту кодировку и сохранить файл в каталоге C:\xampp\htdocs\cp1251.

5.1.3. Методы встраивания PHP-кода

PHP-код встраивается в документ с помощью *дескрипторов*, иногда называемых также *тегами*:

- <?php и ?>:

  ```
  <?php echo "Hello, world\n"; ?>
  ```

  Приведенное выражение можно записать в более компактном виде:

  ```
  <?="Hello, world\n"?>
  ```

  Отключить поддержку этих дескрипторов в PHP 7 нельзя;

- <? и ?>:

  ```
  <? echo "Hello, world\n"; ?>
  ```

  Доступны, только если директива short_open_tag имеет значение On (см. *разд. 4.5*). В пакете XAMPP директива short_open_tag имеет значение Off, поэтому приведенное выражение будет отправлено Web-браузеру без обработки (отобразите исходный HTML-код, чтобы убедиться в этом). При работе с этими дескрипторами следует помнить, что могут возникнуть проблемы при выводе XML-документов, т. к. последовательность <?xml ... ?> будет воспринята как выделение PHP-кода.

Если после закрывающего дескриптора не планируется вывода какого-либо текста, то закрывающий дескриптор можно не указывать вовсе. В этом случае считается, что программа завершается в конце файла. Следовательно, можно написать так:

```
<?php
echo "Hello, world\n";
```

Если файл содержит только PHP-код, то такой вариант встраивания является предпочтительным. Кроме того, он позволит избежать ошибок при выводе с помощью программы различных данных, например изображений. Ведь любой символ (пробел или символ переноса строки) после закрывающего дескриптора будет отправлен Web-браузеру, и мы этого можем даже не заметить. Но такой лишний символ приведет к ошибке.

5.1.4. Комментарии в PHP-сценариях

Комментарии предназначены для вставки пояснений в текст скрипта, и интерпретатор полностью их игнорирует. Внутри комментария может располагаться любой текст, включая инструкции, которые выполнять не следует. Помните, комментарии нужны программисту, а не интерпретатору PHP. Вставка комментариев в код позволит через некоторое время быстро вспомнить предназначение фрагмента кода.

Все, что расположено после // или # до конца строки, в PHP считается *однострочным комментарием*:

```
// Однострочный комментарий
# Однострочный комментарий
```

Однострочный комментарий можно записать после выражения:

```
echo "Hello, world"; // Однострочный комментарий
echo "Hello, world"; # Однострочный комментарий
```

Если символ комментария разместить перед инструкцией, то она не будет выполнена:

```
// echo "Hello, world"; // Инструкция выполнена не будет
```

Если символы `//` расположены внутри кавычек, то они не являются признаком начала комментария:

```
echo "// Это НЕ комментарий!!!";
```

Кроме того, существует *многострочный комментарий*. Он начинается с символов `/*` и заканчивается символами `*/`.

```
/*
Многострочный комментарий
*/
```

То, что комментарий называется многострочным, отнюдь не означает, что он не может располагаться на одной строке:

```
/* Комментарий на одной строке */
```

Следует иметь в виду, что многострочные комментарии не могут быть вложенными, так что при комментировании больших блоков необходимо проверять, что в них не встречается закрывающая комментарий комбинация символов `*/`.

Программа Notepad++ позволяет быстро закомментировать фрагмент кода. Для этого необходимо выделить одну инструкцию (или сразу несколько инструкций) и из контекстного меню выбрать пункт **Вкл./Выкл. Комментарий строки**. В результате в начало каждой выделенной строки будет автоматически вставлен однострочный комментарий.

Если закомментированный блок выделить и повторно выбрать из контекстного меню пункт **Вкл./Выкл. Комментарий строки**, то символы комментариев будут автоматически удалены.

С помощью программы Notepad++ можно также вставить многострочный комментарий. Для этого необходимо выделить несколько инструкций и из контекстного меню выбрать пункт **Закомментировать выделенное**. В результате в начало выделенного фрагмента будут вставлены символы `/*`, а в конец — символы `*/`.

Если выделить закомментированный фрагмент кода и выбрать в контекстном меню пункт **Раскомментировать выделенное**, то символы комментария будут удалены.

5.1.5. Вывод результатов работы скрипта

Вывести результат можно с помощью двух операторов:

- `echo` — мы уже применяли его для вывода строчки `"Hello, world"`:

  ```
  echo "Hello, world";
  ```

Можно вывести сразу несколько строк, указав их через запятую:

```
echo "Строка 1", "Строка 2";
```

- `print` — этот оператор позаимствован из языка Perl:

```
print "Hello, world";
```

> ***Внимание!***
> Оператор `echo` ничего не возвращает, а оператор `print` возвращает число `1`.

Вывести строку или значение переменной можно еще и так:

```
<?="Hello, world\n"?>
<?php $x = 10; ?>
x = <?=$x?>
```

Результат:

```
Hello, world
x = 10
```

Большие блоки текста можно выводить, например, следующим образом:

```
<?php
echo 'Строка1<br>
Строка2<br>
Строка3<br>
';
?>
```

Кроме того, можно воспользоваться синтаксисом, который условно называют «документ здесь»:

```
<?php
echo <<<LABEL
Строка1<br>
Строка2<br>
Строка3<br>
LABEL;
?>
```

В этом примере многострочный текст располагается между метками (`LABEL`):

```
echo <<<LABEL
...
LABEL;
```

Вторая (закрывающая) метка должна обязательно находиться на отдельной строке в самом ее начале. После этой метки должна стоять точка с запятой.

Для форматированного вывода данных можно воспользоваться функцией `printf()`. Формат функции:

```
printf(string $format[, mixed $args[, mixed $...]]) : int
```

В параметре `$format` указывается строка специального формата, внутри которой с помощью спецификаторов задаются правила форматирования. Какие спецификаторы используются, мы рассмотрим при изучении форматирования строк. В параметре `$args` через запятую указываются различные значения. Функция возвращает длину выводимой строки:

```
<?php
echo 10.5125484, "\n";              // 10.5125484
$len = printf("%.2f", 10.5125484);  // 10.51
printf("\n Длина строки равна %d", $len);
// Длина строки равна 5
?>
```

Обратите внимание: при работе функции `printf()` с многобайтовыми кодировками можно получить неожиданный результат. Кроме того, результат выполнения зависит от локали. Настройки локали для разных стран отличаются. Например, в одной стране принято десятичный разделитель вещественных чисел выводить в виде точки, в другой — в виде запятой. Вот пример настройки локали:

```
<?php
printf("%.2f\n", 10.5125484);         // 10.51
setlocale(LC_ALL, 'ru_RU', 'Russian_Russia');
printf("%.2f", 10.5125484);           // 10,51
```

5.1.6. Буферизация вывода

Для ускорения работы осуществляется *буферизация* данных. Иными словами, вначале выводимая строка помещается в память. Когда количество данных достигает определенной величины, данные отправляются Web-браузеру. Для примера выведем 5 строк, но перед выводом каждой строки предпишем интерпретатору «заснуть» на одну секунду:

```
<?php
for ($i = 1; $i < 6; $i++) {
   echo "Строка ", $i, "<br>";
   sleep(1); // "Засыпаем" на 1 секунду
}
```

Результат выполнения этого скрипта мы увидим весь целиком только через 5 секунд. В некоторых случаях необходимо отправлять данные сразу в Web-браузер. Иначе пользователь может подумать, что скрипт завис. Сбросить данные из системного буфера и отправить их пользователю позволяет функция `flush()`, указанная после оператора вывода. Следует заметить, что в некоторых случаях (например, если указано значение в директиве `output_buffering`) необходимо дополнительно вызывать функцию `ob_flush()`:

```
<?php
for ($i = 1; $i < 6; $i++) {
   echo "Строка ", $i, "<br>";
```

```
   if (ob_get_level() > 0) ob_flush();
   flush();    // Сбрасываем системный буфер
   sleep(1);   // "Засыпаем" на 1 секунду
}
```

В этом случае строки будут выводиться сразу, а не все одновременно, как это было в предыдущем примере.

Если предварительно включить неявный сброс данных из системного буфера с помощью функции `ob_implicit_flush()`, то вызывать метод `flush()` не нужно. Следует также заметить, что и сами Web-браузеры могут осуществлять буферизацию. Поэтому количество отправляемых данных должно быть больше размера буфера Web-браузера. Обычно недостающие данные заполняют пробелами, например, с помощью функции `str_pad()`:

```
<?php
ob_implicit_flush();
for ($i = 1; $i < 6; $i++) {
   $input = "Строка $i<br>";
   echo str_pad($input, 4095), "\n";
   if (ob_get_level() > 0) ob_flush();
   sleep(1);   // "Засыпаем" на 1 секунду
}
```

Изменить значение директивы `output_buffering` из программы нельзя. Можно это сделать либо в файле php.ini:

```
output_buffering=Off
```

либо в файлах httpd.conf и .htaccess:

```
php_value output_buffering 0
```

Однако из программы можно сбросить данные из буфера и отключить его с помощью функции `ob_end_flush()`:

```
<?php
echo ini_get('output_buffering'); // 4096
if (ob_get_level() > 0) ob_end_flush();
ob_implicit_flush();
for ($i = 1; $i < 6; $i++) {
   $input = "Строка $i<br>";
   echo str_pad($input, 4095), "\n";
   sleep(1);   // "Засыпаем" на 1 секунду
}
```

Итак, мы познакомились с двумя уровнями буферизации вывода. Во-первых, существует системный буфер вывода (нулевой уровень). Во-вторых, при включенной директиве `output_buffering` будет использоваться дополнительный буфер. Помимо этих уровней, можно создать множество пользовательских вложенных буферов внутри программы. Для этого используются следующие функции:

- `ob_start()` — включает буферизацию вывода. Каждый вызов этой функции создает новый уровень. Если указано значение в директиве `output_buffering`, то функция `ob_start()` первый раз вызывается автоматически. Формат функции:

  ```
  ob_start([callable $output_callback=null[, int $chunk_size=0[,
           int $flags=PHP_OUTPUT_HANDLER_STDFLAGS]]]) : bool
  ```

  В параметре `$output_callback` можно указать функцию, которая будет вызвана при сбросе или очистке буфера. Размер буфера можно задать в параметре `$chunk_size` — если указано значение `0`, то сброс содержимого будет осуществлен перед отключением буфера. Параметр `$flags` позволяет ограничить перечень операций с буфером;
- `ob_get_level()` — возвращает уровень вложенности или значение `0`, если буферизация выключена;
- `ob_get_contents()` — возвращает содержимое текущего буфера;
- `ob_get_length()` — возвращает количество байтов, содержащихся в текущем буфере;
- `ob_get_status([<true | false>])` — возвращает массив с информацией о статусе буфера верхнего уровня, если указано значение `false` или параметр не задан, или о статусе буферов всех уровней, если указано значение `true`;
- `ob_flush()` — сбрасывает данные из текущего буфера;
- `ob_clean()` — удаляет все данные из текущего буфера;
- `ob_end_clean()` — удаляет все данные и отключает текущий буфер;
- `ob_get_clean()` — возвращает содержимое текущего буфера, очищает его, а затем отключает буфер;
- `ob_end_flush()` — сбрасывает данные из текущего буфера и отключает его;
- `ob_get_flush()` — возвращает содержимое текущего буфера, сбрасывает данные, а затем отключает буфер.

Пример использования двух вложенных буферов приведен в листинге 5.7.

Листинг 5.7. Буферизация вывода

```
<?php
// Отключаем буфер, если для директивы
// output_buffering задано значение
if (ob_get_level() > 0) ob_end_flush();

echo ob_get_level(), "\n"; // 0
ob_start();                // Буфер 1
echo "Строка1 \n";
echo ob_get_level(), "\n"; // 1

ob_start();                // Буфер 2
echo "Строка2 \n";         // Браузер эту строку не получит!
```

```
$str = ob_get_contents();  // Сохраняем содержимое буфера 2
ob_clean();                // Очистка буфера 2
echo ob_get_level(), "\n"; // 2
echo "Строка3 \n";
ob_end_flush();            // Сброс и отключение буфера 2

echo ob_get_level(), "\n"; // 1
echo "Строка4 \n";
$str2 = ob_get_flush();    // Сброс и отключение буфера 1
echo ob_get_level(), "\n"; // 0

echo "Содержимое переменной \$str:",  $str,  "\n";
echo "Содержимое переменной \$str2:", $str2, "\n";
```

В параметре `$output_callback` функции `ob_start()` можно указать функцию, которая будет вызвана при сбросе или очистке буфера. Функция принимает строку и должна возвращать новую строку или ту же самую. Давайте внутри функции переведем все символы в верхний регистр (листинг 5.8).

Листинг 5.8. Обработка данных буфера до вывода

```
<?php
ob_start(function($str) {
   return mb_strtoupper($str, 'UTF-8');
});

echo "строка"; // Выведет: СТРОКА
ob_end_flush();
```

> ***Примечание***
>
> Отключать буферизацию или включать неявный сброс данных из системного буфера следует только в том случае, если это действительно вам нужно. Во всех остальных случаях лучше использовать буферизацию, т. к. благодаря ей повышается быстродействие приложения.

5.1.7. Преждевременное завершение выполнения программы

В некоторых случаях может возникнуть условие, при котором дальнейшее выполнение программы лишено смысла. Тогда лучше вывести сообщение об ошибке и прервать выполнение программы досрочно. Для досрочного завершения PHP-сценария предусмотрен оператор `exit`:

```
exit;
exit([<Сообщение или статус завершения>]);
```

Пример:

```
<?php
echo "Строка";
exit;
echo "Эта инструкция выполнена не будет!";
```

Если внутри круглых скобок указана строка, то она будет выведена:

```
<?php
echo "Строка ";
exit("Текст сообщения об ошибке");
echo "Эта инструкция выполнена не будет!";
```

Для досрочного завершения PHP-сценария можно также воспользоваться оператором `die`:

```
die;
die([<Сообщение или статус завершения>]);
```

Если внутри круглых скобок указана строка, то она будет выведена, а если статус завершения, то он не выводится. При нормальном завершении в качестве статуса можно отправить число `0`, а при ошибке — например, число `1`:

```
<?php
echo "Строка";
die(0);
echo "Эта инструкция выполнена не будет!";
```

5.2. Переменные и типы данных

Переменные — это участки памяти, в которых программа хранит данные. Переменная похожа на коробку, в которую можно что-то положить, а через некоторое время достать.

5.2.1. Именование переменных

Каждая переменная должна иметь в программе уникальное имя, состоящее из латинских букв, цифр и знаков подчеркивания. Все имена переменных в PHP начинаются со знака `$`. После этого символа не может стоять цифра.

Правильные имена переменных: `$x`, `$strName`, `$y1`, `$_name`.

Неправильные имена переменных: `y`, `ИмяПеременной`, `$1`, `$ИмяПеременной`.

Последнее имя неправильное в том числе и потому, что в нем используются русские буквы. Хотя на самом деле такой вариант также будет работать, но лучше русские буквы все же не применять.

При указании имени переменной важно учитывать регистр букв: `$strName` и `$strname` — разные переменные.

5.2.2. Типы данных и инициализация переменных

В PHP переменные могут содержать следующие основные типы данных:

- `bool` (или `boolean`) — логический тип данных. Может содержать значения `true` или `false` (регистр не важен);
- `int` (или `integer`) — целые числа;
- `float` (или `double`) — вещественные числа;
- `string` — строка;
- `object` — объекты;
- `array` — массивы;
- `resource` — идентификаторы ресурсов, например, файлов;
- `NULL` — означает, что переменная не содержит значения.

При *инициализации* переменной интерпретатор автоматически относит ее к одному из типов данных. Значение переменной присваивается с помощью оператора `=` так:

```
$b = true;          // bool
$x = 7;             // int
$y = 7.8;           // float
$s1 = "Строка";     // string
$s2 = 'Строка';     // string
$arr = [0, 1];      // array
$n = null;          // NULL
```

PHP в любой момент времени изменяет тип переменной в соответствии с данными, хранящимися в ней:

```
$var = "Строка";    // тип string
$var = 7;           // теперь переменная имеет тип int
```

Функция `gettype(<Имя_переменной>)` возвращает тип данных переменной (листинг 5.9). Для целей отладки можно также использовать функцию `var_dump(<Имя_переменной>)`.

Листинг 5.9. Вывод типа данных переменной

```
<?php
$var = null;
echo gettype($var), "\n"; // NULL
var_dump($var);           // NULL
$var = true;
echo gettype($var), "\n"; // boolean
var_dump($var);           // bool(true)
$var = 7;
echo gettype($var), "\n"; // integer
```

```
var_dump($var);            // int(7)
$var = 7.8;
echo gettype($var), "\n"; // double
var_dump($var);            // float(7.8)
$var = 'Строка';
echo gettype($var), "\n"; // string
var_dump($var);            // string(12) "Строка"
$var = array();
echo gettype($var), "\n"; // array
var_dump($var);            // array(0) {}
```

Кроме того, существуют функции проверки конкретного типа данных:

- `is_bool(<Переменная>)` — возвращает `true`, если переменная имеет тип `bool` (логический тип данных);
- `is_int(<Переменная>)` — возвращает `true`, если переменная имеет тип `int` (целое число);
- `is_integer(<Переменная>)` — возвращает `true`, если переменная имеет тип `int` (целое число);
- `is_float(<Переменная>)` — возвращает `true`, если переменная имеет тип `float` (вещественное число);
- `is_double(<Переменная>)` — возвращает `true`, если переменная имеет тип `float` (вещественное число);
- `is_string(<Переменная>)` — возвращает `true`, если переменная имеет тип `string` (строка);
- `is_array(<Переменная>)` — возвращает `true`, если переменная имеет тип `array` (массив);
- `is_object(<Переменная>)` — возвращает `true`, если переменная имеет тип `object` (объект);
- `is_resource(<Переменная>)` — возвращает `true`, если переменная имеет тип `resource` (ресурс);
- `is_null(<Переменная>)` — возвращает `true`, если переменная имеет значение `NULL`.

Проверить тип переменной можно следующим образом:

```
$var = 7;
if (is_int($var)) {
   echo 'Переменная $var имеет тип int';
}
```

5.2.3. Преобразование и приведение типов

Если функция ожидает один тип, а мы передаем другой, то интерпретатор по умолчанию автоматически пытается выполнить преобразование типов. Для явного *пре-*

образования типов данных можно воспользоваться функцией `settype()`, которая преобразует тип переменной в указанный:

```
settype(<Переменная>, <Тип>)
```

Пример:

```
$var = 1.5;
settype($var, "string");
var_dump($var);            // string(3) "1.5"
settype($var, "int");
var_dump($var);            // int(1)
```

Можно также воспользоваться *приведением типов*. Для этого перед переменной в круглых скобках указывается тип, к которому нужно преобразовать значение переменной.

> **Внимание!**
>
> В отличие от функции `settype()`, приведение типов не меняет тип исходной переменной.

Пример приведения типов:

```
$var = 0;
var_dump($var);         // int(0)
$var2 = (bool)$var;
var_dump($var2);        // bool(false)
```

Значение будет преобразовано в `false` в следующих случаях:

- если число равно `0` или `0.0`;
- если указана пустая строка или строка `"0"`;
- если указан пустой массив;
- если переменная содержит значение `null`.

Все остальные значения будут преобразованы в `true`.

Для преобразования типов можно также воспользоваться функциями `boolval()`, `intval()`, `floatval()`, `doubleval()` и `strval()`:

```
$var = 0;
var_dump($var);         // int(0)
$var2 = boolval($var);
var_dump($var2);        // bool(false)
```

5.2.4. Проверка существования переменной

С помощью оператора `isset(<Переменная>)` можно проверить существование переменной. Если переменная существует, то возвращается значение `true`. Для примера переделаем нашу первую программу так, чтобы она «здоровалась» не со всем миром, а только с нами (листинг 5.10).

Листинг 5.10. Проверка существования переменной

```
<?php
if (isset($_GET['name'])) {
   echo 'Hello, ' . $_GET['name'];
}
else {
   echo 'Введите ваше имя<br>';
   echo '<form action="' . $_SERVER['SCRIPT_NAME'] . '">';
   echo '<input type="text" name="name">';
   echo '<input type="submit" value="OK">';
   echo '</form>';
}
```

При первом запуске программы появится приглашение ввести имя. Вводим свое имя (например, `Николай`) и нажимаем кнопку **OK**. В итоге отобразится приветствие **Hello, Николай**.

Оператор `empty(<Переменная>)` проверяет наличие у переменной непустого, ненулевого значения. Возвращает `true`, если переменная пустая, не существует или имеет нулевое значение. Например, код:

```
if (isset($str)) echo "Существует";
else echo "Нет";
echo "<br>";
if (empty($str)) echo "Пустая";
else echo "Нет";
```

вернет следующие значения:

```
Нет
Пустая
```

А если предварительно инициализировать переменную `$str`, например, так:

```
$str = "Строка";
if (isset($str)) echo "Существует";
else echo "Нет";
echo "<br>";
if (empty($str)) echo "Пустая";
else echo "Нет";
```

то вывод программы будет отображен Web-браузером так:

```
Существует
Нет
```

5.2.5. Удаление переменной

Удалить переменную можно с помощью оператора `unset()`:

```
unset(<Переменная>)
```

Этот оператор необходим, если переменная использовалась при обработке данных большого объема и теперь не нужна. Удаление переменной позволит освободить память компьютера:

```
$str = "Строка";
if (isset($str)) echo "Существует";
else echo "Нет";
unset($str);
echo "<br>";
if (isset($str)) echo "Существует";
else echo "Нет";
```

Вывод программы:

```
Существует
Нет
```

5.2.6. Константы

Константы служат для хранения значений, которые не должны изменяться во время работы программы. Создать константу можно с помощью функции `define()`:

```
define(<Имя константы>, <Значение константы>[, <Регистр>])
```

Необязательный параметр `<Регистр>` может содержать значения `true` или `false`. Если указано `true`, то интерпретатор не будет учитывать регистр символов при поиске константы по ее имени, если же задано `false` или параметр не указан, регистр символов имеет значение.

Обратите внимание: в названии константы принято использовать буквы только в верхнем регистре. Если название константы состоит из нескольких слов, то между словами указывается символ подчеркивания. Это позволяет отличить внутри программы константу от обычной переменной.

После объявления константы ее имя указывается в программе без знака `$`:

```
define('AUTHOR1', 'Николай');
echo AUTHOR1, '<br>';                // Николай
echo author1, '<br><br>';
// Предупреждение о неопределенной константе author1
define('AUTHOR2', 'Сергей', true);
echo AUTHOR2, '<br>';                // Сергей
echo author2, '<br><br>';            // Сергей
define('AUTHOR3', 'Иван', false);
echo AUTHOR3, '<br>';                // Иван
echo author3;
// Предупреждение о неопределенной константе author3
```

Создать константу можно также с помощью ключевого слова `const`:

```
const <Имя константы> = <Значение константы>;
```

Регистр имени константы в этом случае имеет значение:

```
const AUTHOR = 'Николай';
echo AUTHOR, '<br>';                // Николай
echo author;
// Предупреждение о неопределенной константе author
```

Обратите внимание: с помощью ключевого слова `const` нельзя создать константу со значением в виде массива, а вот с помощью функции `define()` можно:

```
define('MY_CONST', array(1, 2, 3));
echo MY_CONST[0]; // 1
```

Кроме того, с помощью ключевого слова `const` нельзя создать константу внутри функции, а вот с помощью функции `define()` можно:

```
function test() {
   define('AUTHOR1', 'Николай');  // OK
   echo AUTHOR1, '<br>';
   // const AUTHOR2 = 'Сергей';   // Ошибка
}
```

Если имя константы нужно составить динамически, то для получения значения константы по ее имени в виде строки можно воспользоваться функцией `constant()`:

```
define('MY_CONST', 3);
echo constant('MY_CONST'); // 3
```

Для проверки существования константы служит функция `defined(<Имя константы>)`. Функция возвращает `true`, если константа объявлена:

```
define('AUTHOR', 'Николай');
if (defined('AUTHOR')) echo 'Объявлена';
else echo 'Не объявлена';
```

В PHP существуют встроенные константы, например:

- `__FILE__` (до и после два символа подчеркивания) — содержит имя файла с программой;
- `__LINE__` (до и после два символа подчеркивания) — содержит номер строки, которую обрабатывает интерпретатор в данный момент;
- `PHP_OS` — содержит имя и версию операционной системы;
- `PHP_VERSION` — содержит версию PHP.

Пример:

```
<?php
echo __FILE__ . "<br>";
echo __LINE__ . "<br>";
echo PHP_OS . "<br>";
echo PHP_VERSION . "<br>";
```

В итоге получим HTML-код, отображаемый так:

```
C:\xampp\htdocs\index.php
3
WINNT
7.2.0
```

Получить полный список всех констант позволяет функция `get_defined_constants()`:

```
print_r(get_defined_constants());
```

Список очень большой, поэтому приводить его мы здесь не станем — запустите эту инструкцию и посмотрите результат в исходном HTML-коде.

5.2.7. Переменные окружения

Создадим сценарий, состоящий всего из двух строк:

```
<?php
$var = 10;
```

А теперь вопрос: сколько переменных доступно сценарию? Думаете, одна `$var`? Давайте перепишем нашу программу и добавим одну строчку:

```
<?php
$var = 10;
echo $_SERVER['DOCUMENT_ROOT'];
```

В результате работы скрипта в окне Web-браузера отобразится следующая строка:

```
C:/xampp/htdocs
```

Откуда же взялась переменная `$_SERVER['DOCUMENT_ROOT']`? Ведь мы ее не создавали! Ответ на этот вопрос достаточно прост — эта переменная была автоматически создана интерпретатором. Такая переменная называется *переменной окружения*.

Рассмотренная нами ранее переменная окружения `$_SERVER['DOCUMENT_ROOT']` представляет собой элемент массива `$_SERVER`. Это весьма примечательный массив — он доступен не только в глобальной области видимости, но в любой части скрипта, из-за чего и носит название *суперглобального*.

Приведем суперглобальные массивы:

- `$_GET` — массив переменных, переданных посредством метода `GET`;
- `$_POST` — массив переменных, переданных посредством метода `POST`;
- `$_SERVER` — массив переменных среды сервера;
- `$_FILES` — массив переменных, определяющих отправленные через форму файлы;
- `$_COOKIE` — массив cookies-переменных;
- `$_ENV` — массив переменных, определяющих конфигурацию среды;
- `$_REQUEST` — массив всех переменных, вводимых пользователем. Этот массив зависит от значения директивы `request_order`.

Вывести значения всех переменных окружения позволяет код из листинга 5.11.

Листинг 5.11. Переменные окружения

```
<pre>
<?php
echo "\$_GET\n";
print_r($_GET);
echo "\n\$_POST\n";
print_r($_POST);
echo "\n\$_SERVER\n";
print_r($_SERVER);
echo "\n\$_FILES\n";
print_r($_FILES);
echo "\n\$_COOKIE\n";
print_r($_COOKIE);
echo "\n\$_ENV\n";
print_r($_ENV);
echo "\n\$_REQUEST\n";
print_r($_REQUEST);
?>
</pre>
```

Список очень большой, поэтому приводить его здесь мы не станем — запустите код и посмотрите результат самостоятельно.

Рассмотрим наиболее часто используемые переменные окружения:

- `$_SERVER['DOCUMENT_ROOT']` — путь к корневому каталогу сервера;
- `$_SERVER['REMOTE_ADDR']` — IP-адрес клиента, запрашивающего ресурс;
- `$_SERVER['REMOTE_USER']` — имя пользователя, прошедшего аутентификацию;
- `$_SERVER['QUERY_STRING']` — строка переданных серверу параметров;
- `$_SERVER['HTTP_USER_AGENT']` — название и версия Web-браузера клиента;
- `$_SERVER['HTTP_REFERER']` — URL-адрес, с которого пользователь перешел на наш сайт;
- `$_SERVER['REQUEST_METHOD']` — метод передачи информации (`GET` или `POST`).

Прежде чем использовать переменные окружения, необходимо проверить существование переменной с помощью оператора `isset()`:

```
if (isset($_SERVER['HTTP_REFERER'])) {
   echo $_SERVER['HTTP_REFERER'];
}
```

Предположим, что пользователь заполнил форму с одним текстовым полем, имеющим имя `text1` (`name="text1"`). При передаче данных методом `GET` сервер сформирует следующие переменные:

```
$_GET['text1']
$_REQUEST['text1']
```

Если передача формы осуществлялась методом `POST`, то сервер сформирует другие переменные:

```
$_POST['text1']
$_REQUEST['text1']
```

Мы можем извлечь ее значение и присвоить обычной переменной PHP:

```
if (isset($_GET['text1'])) $text1 = $_GET['text1'];
else $text1 = '';
```

или

```
if (isset($_POST['text1'])) $text1 = $_POST['text1'];
else $text1 = '';
```

Аналогичную операцию проще выполнить с помощью оператора `??`:

```
$text1 = $_GET['text1'] ?? '';
```

или

```
$text1 = $_POST['text1'] ?? '';
```

Если переменная окружения существует, то переменная `$text1` получит ее значение, в противном случае переменной `$text1` будет присвоена пустая строка.

Остальные переменные окружения используются реже, но по названиям их предназначение интуитивно понятно. В дальнейшем мы еще не раз будем возвращаться к переменным окружения.

Управлять порядком обработки суперглобальных массивов позволяет директива `variables_order` в файле php.ini. В пакете XAMPP директива имеет следующее значение:

```
variables_order="GPCS"
```

Буквы, указанные внутри строки, это первые буквы в именах `GET`, `POST`, `COOKIE`, `SERVER` и `ENV`. В этом примере отсутствует буква `E`, т. к. буква `S` всегда эквивалента комбинации `ES`. Если какая-либо буква не указана, то соответствующий массив не создается.

Более интересен порядок букв в директиве `request_order`. Эта директива задает порядок заполнения массива `$_REQUEST`. В пакете XAMPP директива имеет следующее значение:

```
request_order="GP"
```

Можно указать буквы `G`, `P` и `C`. Обратите внимание: буква `C` по умолчанию не указана, следовательно, cookies-переменные в массив `$_REQUEST` добавлены не будут. Добавление производится слева направо, при этом старые значения будут перезаписаны новыми. Иными словами, если указан порядок `GP`, то вначале будет создана переменная со значением, переданным методом `GET`, а затем — переменная со зна-

чением, переданным методом POST. Если переменные имеют одинаковое имя, то мы получим значение, переданное методом POST, а не GET. Если же порядок букв изменить на противоположный, то, наоборот, мы получим значение, переданное методом GET, а не POST.

5.2.8. Массив *$GLOBALS*

Все глобальные переменные, определенные внутри программы, доступны через массив $GLOBALS. Чтобы обратиться к переменной через этот массив, нужно указать ее имя в виде строки внутри квадратных скобок. Благодаря тому, что имя переменной указывается в виде строки, можно создавать имя динамически во время выполнения программы:

```
<?php
$x = 10;
echo $GLOBALS['x'];    // 10
```

Мы можем не только получить значение переменной через массив $GLOBALS, но и создать глобальную переменную:

```
<?php
$GLOBALS['x'] = 10;
echo $x;               // 10
```

5.2.9. Вывод значений переменных

Для вывода значений переменных можно воспользоваться оператором echo:

```
$x = 15;
echo $x;
echo "\n Значение переменной равно ", $x;
```

Имя переменной допускается указывать внутри строки. Если строка заключена в двойные кавычки, то мы получим значение переменной, а если в одинарные — то ее имя. Если нужно получить имя переменной внутри строки в двойных кавычках, то перед знаком $ следует указать символ \:

```
$x = 15;
echo 'Значение переменной $x равно ', $x, "\n";
echo "Значение переменной \$x равно ", $x, "\n";
echo "Значение переменной \$x равно $x";
```

В результате получим три строки:

```
Значение переменной $x равно 15
```

Чтобы явным образом выделить имя переменной внутри строки и исключить слияние имени с другими буквами, следует использовать фигурные скобки:

```
$x = 15;
echo "{$x} ${x}"; // 15 15
```

Для вывода массивов оператор `echo` не подходит. При его использовании мы получим предупреждающее сообщение и просто слово `Array` вместо значений элементов массива. Вывести значения элементов массива, а также любых других данных позволяет функция `print_r()`:

```
$x = 15;
print_r($x);       // 15
$arr = [1, 2, 3];
print_r($arr);     // Array ( [0] => 1 [1] => 2 [2] => 3 )
```

Для целей отладки и поиска ошибок будет полезно не только получить значение переменной, но и информацию о типе данных. В этом случае следует воспользоваться функцией `var_dump()`:

```
$x = 15;
var_dump($x);       // int(15)
$arr = [1, 2, 3];
var_dump($arr);
// array(3) { [0]=> int(1) [1]=> int(2) [2]=> int(3) }
```

Для вставки значения переменной в HTML-код удобно воспользоваться PHP-дескрипторами `<?=...?>`. Однако следует учитывать, что при необходимости нужно заменять специальные символы их HTML-эквивалентами с помощью функции `htmlspecialchars()`:

```
<?php
$x = 15;
$txt = "<>";
$txt2 = htmlspecialchars($txt, ENT_COMPAT | ENT_HTML5,
                         'UTF-8');
?>
<input type="text" name="x" value="<?=$x?>">
<input type="text" name="txt" value="<?=$txt2?>">
```

Результат:

```
<input type="text" name="x" value="15">
<input type="text" name="txt" value="&lt;&gt;">
```

5.2.10. Ссылки

Ссылку на значение одной переменной можно сохранить в другой переменной — достаточно перед именем переменной указать символ `&`. После этого любое изменение значения в одной переменной приведет к изменению значения в другой переменной:

```
$x = 15;
$y = &$x;
var_dump($x); // int(15)
var_dump($y); // int(15)
```

```
$y = 20;
var_dump($x); // int(20)
var_dump($y); // int(20)
$x = 33;
var_dump($x); // int(33)
var_dump($y); // int(33)
```

Для разрыва связи следует воспользоваться оператором `unset()`:

```
$x = 15;
$y = &$x;
var_dump($x); // int(15)
var_dump($y); // int(15)
$y = 20;
var_dump($x); // int(20)
var_dump($y); // int(20)
unset($y);
$y = 33;
var_dump($x); // int(20)
var_dump($y); // int(33)
```

Помимо обычных ссылок (которые иногда называют «жесткими») в PHP существуют *символические ссылки* (другое их название: «переменные переменных»). Символическая ссылка — это строковая переменная, хранящая имя другой переменной. Чтобы получить значение такой переменной или изменить его, нужно указать два символа `$`:

```
$a = "test";
$$a = 50;
var_dump($a);    // string(4) "test"
var_dump(${$a}); // int(50)
var_dump($test); // int(50)
```

5.3. Операторы

Операторы позволяют выполнить с данными определенные действия. Например, операторы присваивания служат для сохранения данных в переменной, математические операторы предназначены для арифметических вычислений, а условные операторы позволяют в зависимости от значения логического выражения выполнить отдельный участок программы или, наоборот, не выполнять его. Рассмотрим операторы, доступные в PHP, более подробно.

5.3.1. Математические операторы

Производить арифметические вычисления позволяют следующие операторы:

- `+` — сложение:

  ```
  echo 10 + 15;          // 25
  ```

- `-` — вычитание:

```
echo 35 - 15;           // 20
```

- `-` — унарный минус:

```
$x = 10;
echo -$x;               // -10
```

- `*` — умножение:

```
echo 25 * 2;            // 50
```

- `/` — деление. Если в выражении участвуют только целые числа, и деление выполняется без остатка, то результатом будет целое число. В противном случае возвращается вещественное число:

```
echo 10 / 2;            // 5
echo 10 / 3;            // 3.3333333333333
echo 10.0 / 3;          // 3.3333333333333
echo 10.0 / 3.0;        // 3.3333333333333
```

  Если нужно выполнить целочисленное деление, то следует воспользоваться функцией `intdiv()`:

```
echo intdiv(10, 3);     // 3
```

  Деление числа на `0` приведет к значению плюс или минус `INF` (бесконечность), а деление вещественного числа `0.0` на `0` — к значению `NaN` (нет числа):

```
echo 10.0 / 0;          // INF + Warning:  Division by zero
echo -10.0 / 0;         // -INF + Warning:  Division by zero
echo 0.0 / 0;           // NAN + Warning:  Division by zero
```

- `%` — остаток от деления:

```
echo 10 % 2;            // 0
echo 10 % 3;            // 1
echo 10 % 4;            // 2
echo 10 % 6;            // 4
```

- `**` — возведение в степень:

```
echo 10 ** 2;           // 100
echo 3 ** 3;            // 27
```

- `++` — оператор инкремента. Увеличивает значение переменной на `1`:

```
$x = 10;
$x++;                   // Эквивалентно $x = $x + 1;
echo $x;                // 11
```

- `--` — оператор декремента. Уменьшает значение переменной на `1`:

```
$x = 10;
$x--;                   // Эквивалентно $x = $x - 1;
echo $x;                // 9
```

Операторы инкремента и декремента можно записывать в постфиксной или префиксной формах:

```
$x++; $x--;      // Постфиксная форма
++$x; --$x;      // Префиксная форма
```

При постфиксной форме (`$x++`) сначала возвращается значение переменной, а потом выполняется операция, а при префиксной форме (`++$x`) — вначале осуществляется операция и только потом возвращается значение. Продемонстрируем это на примере (листинг 5.12).

Листинг 5.12. Постфиксная и префиксная формы

```
<?php
$x = 5;
$y = $x++; // $y = 5, $x = 6
echo "<b>Постфиксная форма (\$y = \$x++;):</b><br>\n";
echo "\$y = $y <br>\$x = $x <br>\n";
$x = 5;
$y = ++$x; // $y = 6, $x = 6
echo "<b>Префиксная форма (\$y = ++\$x;):</b><br>\n";
echo "\$y = $y <br>\$x = $x";
```

В итоге получим следующий результат:

```
Постфиксная форма ($y = $x++;):
$y = 5
$x = 6
Префиксная форма ($y = ++$x;):
$y = 6
$x = 6
```

Если операторы инкремента и декремента используются в сложных выражениях, то понять, каким будет результат выполнения выражения, становится сложно. Например, каким будет значение переменной `$y` после выполнения этих инструкций?

```
$x = 5; $y = 0;
$y = ++$x + ++$x + ++$y;
echo $y;                    // ???
```

Чтобы облегчить жизнь себе и всем другим программистам, которые будут разбираться в программе, операторы инкремента и декремента лучше использовать отдельно от других операторов.

5.3.2. Побитовые операторы

Побитовые операторы предназначены для манипуляции отдельными битами. Язык PHP поддерживает следующие побитовые операторы:

- `~` — двоичная инверсия. Значение каждого бита заменяется на противоположное:

```
$x = 100;
printf("%032b\n", $x); // 00000000000000000000000001100100
$x = ~$x;
printf("%032b\n", $x); // 11111111111111111111111110011011
```

- `&` — двоичное И:

```
$x = 100; $y = 75;
printf("%032b\n", $x); // 00000000000000000000000001100100
printf("%032b\n", $y); // 00000000000000000000000001001011
$z = $x & $y;
printf("%032b\n", $z); // 00000000000000000000000001000000
```

- `|` — двоичное ИЛИ:

```
$x = 100; $y = 75;
printf("%032b\n", $x); // 00000000000000000000000001100100
printf("%032b\n", $y); // 00000000000000000000000001001011
$z = $x | $y;
printf("%032b\n", $z); // 00000000000000000000000001101111
```

- `^` — двоичное исключающее ИЛИ:

```
$x = 100; $y = 250;
printf("%032b\n", $x); // 00000000000000000000000001100100
printf("%032b\n", $y); // 00000000000000000000000011111010
$z = $x ^ $y;
printf("%032b\n", $z); // 00000000000000000000000010011110
```

- `<<` — сдвиг влево — сдвигает двоичное представление числа влево на один или более разрядов и заполняет разряды справа нулями:

```
$x = 100;
printf("%032b\n", $x); // 00000000000000000000000001100100
$x = $x << 1;
printf("%032b\n", $x); // 00000000000000000000000011001000
$x = $x << 1;
printf("%032b\n", $x); // 00000000000000000000000110010000
$x = $x << 2;
printf("%032b\n", $x); // 00000000000000000000011001000000
```

- `>>` — сдвиг вправо — сдвигает двоичное представление числа вправо на один или более разрядов и заполняет старшие разряды содержимым самого старшего разряда:

```
$x = 100;
printf("%032b\n", $x); // 00000000000000000000000001100100
$x = $x >> 1;
printf("%032b\n", $x); // 00000000000000000000000000110010
$x = $x >> 1;
printf("%032b\n", $x); // 00000000000000000000000000011001
$x = $x >> 2;
printf("%032b\n", $x); // 00000000000000000000000000000110
```

Если число отрицательное, то разряды слева заполняются единицами:

```
$x = -127;
printf("%032b\n", $x); // 11111111111111111111111110000001
$x = $x >> 1;
printf("%032b\n", $x); // 11111111111111111111111111000000
$x = $x >> 1;
printf("%032b\n", $x); // 11111111111111111111111111100000
$x = $x >> 2;
printf("%032b\n", $x); // 11111111111111111111111111111000
```

5.3.3. Операторы присваивания

Операторы присваивания предназначены для сохранения значения в переменной. Приведем операторы присваивания, доступные в языке PHP:

- `=` — присваивает переменной значение. Обратите внимание на то, что хотя оператор похож на математический знак равенства, смысл у него совершенно иной. Справа от оператора присваивания может располагаться константа или сложное выражение. Слева от оператора присваивания может располагаться только переменная или инструкция `list()`:

```
$x = 5;
echo $x;       // 5
$arr = ['Ноль', 'Один', 'Два', 'Три'];
list($var1, $var2, $var3, $var4) = $arr;
```

- `+=` — увеличивает значение переменной на указанную величину:

```
$x = 10;
$x += 5;       // Эквивалентно $x = $x + 5;
echo $x;       // 15
```

- `-=` — уменьшает значение переменной на указанную величину:

```
$x = 10;
$x -= 5;       // Эквивалентно $x = $x - 5;
echo $x;       // 5
```

- `*=` — умножает значение переменной на указанную величину:

```
$x = 10;
$x *= 5;       // Эквивалентно $x = $x * 5;
echo $x;       // 50
```

- `/=` — делит значение переменной на указанную величину:

```
$x = 10;
$x /= 5;       // Эквивалентно $x = $x / 5;
echo $x;       // 2
```

- `%=` — делит значение переменной на указанную величину и возвращает остаток:

```
$x = 10;
$x %= 5;       // Эквивалентно $x = $x % 5;
echo $x;       // 0
```

- `**=` — возведение в степень:

```
$x = 10;
$x **= 2;      // Эквивалентно $x = $x ** 2;
echo $x;       // 100
```

- `&=`, `|=`, `^=`, `<<=` и `>>=` — побитовые операторы с присваиванием:

```
$x = 100; $y = 75;
$x |= $y;              // Эквивалентно $x = $x | $y;
printf("%032b\n", $x); // 00000000000000000000000001101111
```

5.3.4. Операторы сравнения

Операторы сравнения входят в состав логических выражений. Приведем их список:

- `==` — равно;
- `===` — строго равно;
- `!=` — не равно;
- `<>` — не равно;
- `!==` — строго не равно;
- `<` — меньше;
- `>` — больше;
- `<=` — меньше или равно;
- `>=` — больше или равно.

Логические выражения возвращают только два значения: `true` (истина) или `false` (ложь). Вот пример вывода значения логического выражения:

```
var_dump( 10 == 10 ); // bool(true)
var_dump( 10 == 5  ); // bool(false)
var_dump( 10 != 5  ); // bool(true)
var_dump( 10 <> 5  ); // bool(true)
var_dump( 10 <  5  ); // bool(false)
var_dump( 10 >  5  ); // bool(true)
var_dump( 10 <= 5  ); // bool(false)
var_dump( 10 >= 5  ); // bool(true)
```

В чем отличие оператора `==` (равно) от оператора `===` (строго равно)? Если используется оператор `==`, интерпретатор пытается преобразовать разные типы данных к одному и лишь затем сравнивает их. Оператор `===`, встретив данные разных типов, сразу возвращает `false`:

```
var_dump(1 ==  "1"); // bool(true)
var_dump(1 === "1"); // bool(false)
```

Значение логического выражения можно инвертировать с помощью оператора `!`:

```
$var1 = 5;
$var2 = 5;
```

```
var_dump(   $var1 == $var2  ); // bool(true)
var_dump( !($var1 == $var2) ); // bool(false)
```

Если переменные `$var1` и `$var2` равны, то возвращается значение `true`, но так как перед выражением стоит оператор `!`, выражение вернет `false`.

Можно несколько логических выражений объединить в одно большое с помощью следующих операторов:

- `&&` или `and` — логическое `И`. Логическое выражение вернет `true` только в случае, если оба подвыражения вернут `true`:

```
var_dump( (10 == 10) &&  (5 != 3) ); // bool(true)
var_dump( (10 == 10) &&  (5 == 3) ); // bool(false)
var_dump( (10 == 10) and (5 != 3) ); // bool(true)
var_dump( (10 == 10) and (5 == 3) ); // bool(false)
```

- `||` или `or` — логическое `ИЛИ`. Логическое выражение вернет `true`, если хотя бы одно из подвыражений вернет `true`:

```
var_dump( (10 == 10) || (5 != 3) );  // bool(true)
var_dump( (10 == 10) || (5 == 3) );  // bool(true)
var_dump( (10 == 10) or (5 != 3) );  // bool(true)
var_dump( (10 == 10) or (5 == 3) );  // bool(true)
```

Если первое подвыражение вернет значение `true`, то второе подвыражение даже не будет вычисляться. Например, в этом выражении деление на `0` никогда не будет выполнено (следовательно, и ошибки не возникнет):

```
var_dump( (10 == 10) || ((10 / 0) > 0) ); // bool(true)
```

- `xor` — логическое исключающее `ИЛИ`. Логическое выражение вернет `true`, если хотя бы одно из подвыражений вернет `true`, но не оба:

```
var_dump( (10 == 10) xor (5 != 3) );  // bool(false)
var_dump( (10 == 10) xor (5 == 3) );  // bool(true)
```

> **Внимание!**
> Приоритет операторов `&&` и `||` выше, чем приоритет операторов `and` и `or`.

5.3.5. Оператор <=>

Оператор `<=>` сравнивает значения двух переменных: `$a` и `$b`. Возвращает отрицательное число, положительное число или число, равное нулю, когда `$a` меньше, больше или равно `$b` соответственно:

```
echo 1 <=> 2; // -1
echo 3 <=> 2; // 1
echo 2 <=> 2; // 0
```

Оператор `<=>` заменяет следующую конструкцию:

```
$a = 1;
$b = 2;
```

```
if ($a < $b)      $x = -1;
elseif ($a > $b) $x = 1;
else              $x = 0;
echo $x;
```

Обратите внимание: оператор <=> выполняет нестрогое сравнение, при котором строка может быть преобразована в число. В итоге будут сравниваться числа, а не строки. Учитывайте это при сравнении строк:

```
var_dump("10 a" <=> 10);      // int(0)
var_dump("10" <=> "010");     // int(0)
```

5.3.6. Оператор *??*

Если переменная, указанная слева от оператора `??`, имеет значение `null`, то возвращается значение, указанное справа от оператора. В противном случае возвращается значение переменной:

```
$x = $y ?? 5;
var_dump( $x ); // int(5)
$y = 10;
$x = $y ?? 5;
var_dump( $x ); // int(10)
$y = null;
$x = $y ?? 5;
var_dump( $x ); // int(5)
```

Обратите внимание: если переменная не определена, то никакого предупреждающего сообщения выдано не будет. Следовательно, оператор `??` можно использовать вместо следующей конструкции:

```
if (isset($y)) $x = $y;
else $x = 5;
```

Согласитесь, что эта инструкция выглядит значительно нагляднее:

```
$x = $y ?? 5;
```

5.3.7. Приоритет выполнения операторов

В какой последовательности будет вычисляться приведенное далее выражение?

```
$x = 5 + 10 * 3 / 2;
echo $x;   // 20
```

Это зависит от приоритета выполнения операторов. В данном случае последовательность вычисления выражения будет следующей:

1. Число `10` будет умножено на `3`, т. к. приоритет оператора умножения выше приоритета оператора сложения.
2. Полученное значение будет поделено на `2`, поскольку приоритет оператора деления равен приоритету оператора умножения (а операторы с равными приоритетами выполняются слева направо), но выше чем у оператора сложения.

3. К полученному значению будет прибавлено число 5, т. к. оператор присваивания = имеет наименьший приоритет.
4. Значение будет присвоено переменной $x.

С помощью скобок можно изменить последовательность вычисления выражения:

```
$x = (5 + 10) * 3 / 2;
echo $x;   // 22.5
```

Теперь порядок вычислений будет другим:

1. К числу 5 будет прибавлено 10.
2. Полученное значение будет умножено на 3.
3. Полученное значение будет поделено на 2.
4. Значение будет присвоено переменной $x.

Приведем рассмотренные операторы в порядке убывания приоритета:

- ** — возведение в степень.
- ++, --, ~, (<Тип>) — инкремент, декремент, двоичная инверсия, приведение типов.
- ! — логическое отрицание.
- *, /, % — умножение, деление, остаток от деления.
- +, -, . — сложение, вычитание, конкатенация строк.
- <<, >> — двоичный сдвиг.
- <, <=, >, >= — операторы сравнения.
- ==, !=, ===, !==, <>, <=> — операторы сравнения.
- & — двоичное И.
- ^ — двоичное исключающее ИЛИ.
- | — двоичное ИЛИ.
- && — объединение логических операторов.
- || — объединение логических операторов.
- ?? — проверка переменной на значение null.
- =, +=, -=, *=, **=, /=, %=, &=, |=, ^=, <<= и >>= — присваивание.
- and — объединение логических операторов.
- xor — объединение логических операторов.
- or — объединение логических операторов.

5.3.8. Преобразование типов данных

Что получится, если к числу прибавить строку?

```
$str = "5";                  // Строка
$number = 3;                 // Число
```

```
$var1 = $number + $str; // Переменная содержит число 8
$var2 = $str + $number; // Переменная содержит число 8
```

Результат будет абсолютно не таким, как в JavaScript, поскольку оператор + в PHP не используется для конкатенации строк. В этом случае интерпретатор попытается преобразовать переменные к одному типу данных, а затем выполнить операцию. В нашем примере переменная `$str`, имеющая тип `string` (строка), будет преобразована к типу `int` (число), а затем будет выполнено сложение двух чисел.

Но что произойдет, если строку невозможно преобразовать в число?

```
$str = "Привет";        // Строка
$number = 3;            // Число
$var1 = $number + $str; // Переменная содержит число 3
$var2 = $str + $number; // Переменная содержит число 3
```

Как видно из примера, строка, не содержащая числа, преобразуется к числу `0`. При этом будет выведено предупреждающее сообщение о невозможности преобразования типов:

```
Warning: A non-numeric value encountered
```

А что будет, если из числа вычесть строку, число умножить на строку или число разделить на строку?

```
$number = 15;           // Число
$str = "5";             // Строка
$var1 = $number - $str; // Переменная содержит число 10
$var2 = $number * $str; // Переменная содержит число 75
$var3 = $number / $str; // Переменная содержит число 3
```

Как можно видеть, интерпретатор попытается преобразовать строку в число, а затем вычислить выражение. В какой последовательности будут указаны число и строка, не важно:

```
$var4 = $str * $number; // Переменная все равно содержит число 75
```

С одной стороны, хорошо, что интерпретатор выполняет преобразование типов данных за нас. Но с другой стороны, можно получить результат, который вовсе не планировался. Поэтому лучше оперировать переменными одного типа, а при необходимости преобразования типов делать это самостоятельно:

```
$str = "5";                     // Строка
$number = 3;                    // Число
$var1 = $number + intval($str); // Переменная содержит число 8
$var2 = $number + (int)$str;    // Переменная содержит число 8
var_dump($var1);                // int(8)
var_dump($var1);                // int(8)
```

5.3.9. Оператор ветвления *if*

Оператор ветвления уже встречался ранее в наших примерах — в частности, так мы определяли факт существования переменной. Так как оператор `isset()` при суще-

ствовании переменной возвращает значение `true`, то это условие можно проверить, используя оператор ветвления `if...else`:

```
if (isset($_GET['name'])) {
   echo 'Hello, ' . $_GET['name'];
}
else {
   echo 'Введите ваше имя<br>';
   echo '<form action="' . $_SERVER['SCRIPT_NAME'] . '">';
   echo '<input type="text" name="name">';
   echo '<input type="submit" value="OK">';
   echo '</form>';
}
```

Обратите внимание, что логическое выражение не содержит операторов сравнения:

```
if (isset($_GET['name'])) {
```

Такая запись эквивалентна следующей:

```
if (isset($_GET['name']) == true) {
```

Проверка на равенство выражения значению `true` выполняется по умолчанию.

Оператор ветвления `if...else` имеет следующий формат:

```
if (<Логическое выражение>) {
   <Блок, выполняемый, если условие истинно>
}
[elseif (<Логическое выражение>) {
   <Блок, выполняемый, если условие истинно>
}]
[else {
   <Блок, выполняемый, если все условия ложны>
}]
```

Если условие в `if` истинно, то выполняются инструкции из этого блока, а все остальные условия пропускаются. Если условие в `if` ложно, то проверяется условие в первом `elseif`. Если условие в `elseif` истинно, то выполняются инструкции из этого блока, а все остальные условия пропускаются. Если условие в первом `elseif` ложно, то точно так же проверяются остальные условия. Если все условия ложны, то выполняется блок `else`. Блоков `elseif` может быть несколько с разными логическими выражениями, а вот блок `else` может быть только один. Блоки `elseif` и `else` являются необязательными.

Напишем программу, которая проверяет, является ли введенное пользователем число четным или нет (листинг 5.13). После проверки выводится соответствующее сообщение.

Листинг 5.13. Проверка числа на четность

```
<b>Проверка числа на четность</b><br><br>
Введите число<br>
```

```
<form action="<?=$_SERVER['SCRIPT_NAME']?>">
<input type="text" name="var">
<input type="submit" value="OK">
</form><br>
<?php
if (isset($_GET['var'])) {
   $var = $_GET['var'];
   if (preg_match('/^[0-9]+$/su', $var)) {
      // Преобразуем тип string (строка) в int (число)
      $var = intval($var, 10);
      if ( ($var % 2) == 0 ) {
         echo $var . ' — четное число';
      }
      else {
         echo $var . ' — нечетное число';
      }
   }
   else echo 'Необходимо ввести число';
}
?>
```

Как видно из примера, один условный оператор можно вложить в другой. Кроме того, если блок состоит из одного выражения, фигурные скобки можно не указывать:

```
if ( ($var % 2) == 0 ) echo $var . ' — четное число';
else echo $var . ' — нечетное число';
```

Блок `else` может отсутствовать:

```
if ( ($var % 2) == 0 ) echo $var . ' — четное число';
```

Кроме того, оператор `if...else` позволяет проверить сразу несколько условий. Рассмотрим это на примере (листинг 5.14).

Листинг 5.14. Проверка введенного значения

```
<b>Какой операционной системой вы пользуетесь?</b><br><br>
<form action="<?=$_SERVER['SCRIPT_NAME']?>">
<select name="os">
<option value="0" selected>Не выбрано</option>
<option value="1">Windows XP</option>
<option value="2">Windows 7</option>
<option value="3">Windows 8</option>
<option value="4">Windows 10</option>
<option value="5">Другая</option>
</select>
<input type="submit" value="Выбрал">
</form>
```

```
<?php
if (isset($_GET['os'])) {
   $os = $_GET['os'];
   if ($os == '1')     echo 'Вы выбрали — Windows XP';
   elseif ($os == '2') echo 'Вы выбрали — Windows 7';
   elseif ($os == '3') echo 'Вы выбрали — Windows 8';
   elseif ($os == '4') echo 'Вы выбрали — Windows 10';
   elseif ($os == '5') echo 'Вы выбрали - Другая';
   elseif ($os == '0') echo 'Вы не выбрали операционную систему';
   else echo 'Мы не смогли определить вашу операционную систему';
}
?>
```

С помощью оператора `elseif` мы можем определить выбранное в списке значение и вывести соответствующее сообщение.

Существует также следующий формат оператора ветвления `if...else`:

```
<?php if (<Логическое выражение>): ?>
   <Текст, выводимый, если условие истинно>
[<?php elseif (<Логическое выражение>): ?>
   <Текст, выводимый, если условие истинно>]
[<?php else: ?>
   <Текст, выводимый, если все условия ложны>]
<?php endif; ?>
```

Этот формат удобно использовать для вывода HTML-кода:

```
<?php if (isset($_GET['name'])): ?>
Hello, <?=$_GET['name']?>
<?php else: ?>
Введите ваше имя<br>
<form action="<?=$_SERVER['SCRIPT_NAME']?>">
<input type="text" name="name">
<input type="submit" value="OK">
</form>
<?php endif; ?>
```

5.3.10. Оператор *?:*

Оператор `?:` имеет следующий формат:

```
<Переменная> = (<Логическое выражение>) ? <Выражение если Истина> :
                                          <Выражение если Ложь>;
```

Если логическое выражение возвращает значение `true`, то выполняется выражение, расположенное после вопросительного знака. Если логическое выражение возвращает значение `false`, то выполняется выражение, расположенное после двоеточия. Результат выполнения выражений становится результатом выполнения оператора.

Перепишем нашу программу проверки числа на четность (листинг 5.13) и используем оператор `?:` вместо `if...else` (листинг 5.15).

Листинг 5.15. Использование оператора `?:`

```
<b>Проверка числа на четность</b><br><br>
Введите число<br>
<form action="<?=$_SERVER['SCRIPT_NAME']?>">
<input type="text" name="var">
<input type="submit" value="OK">
</form><br>
<?php
if (isset($_GET['var'])) {
   $var = $_GET['var'];
   if (preg_match('/^[0-9]+$/su', $var)) {
      // Преобразуем тип string (строка) в int (число)
      $var = intval($var, 10);
      echo ( ($var % 2) == 0) ? $var . ' — четное число' :
                                $var . ' — нечетное число';
   }
   else echo 'Необходимо ввести число';
}
?>
```

Рассмотрим еще один пример. Предположим, необходимо вывести сообщение при возникновении определенного условия. Если попробовать вывести его так:

```
$var = 5;
($var == 5) ? echo 'Равно' : echo 'Не равно'; // Ошибка
```

то возникнет ошибка, потому что подвыражение должно возвращать какое-либо значение, а оператор `echo` ничего не возвращает. Обойти эту ошибку можно заменой оператора `echo` на `print`:

```
$var = 5;
($var == 5) ? print 'Равно' : print 'Не равно';
```

Средний параметр можно не указывать:

```
$var = 6;
($var == 5) ? : print 'Не равно';
```

5.3.11. Оператор выбора *switch*

Оператор выбора `switch` имеет следующий формат:

```
switch (<Переменная или выражение>) {
   case <Значение 1>:
      <Выражение 1>;
      break;
```

```
[  case <Значение 2>:
      <Выражение 2>;
      break;
...]
[  default:
      <Выражение>;]
}
```

Вместо логического выражения оператор `switch` принимает переменную или выражение (вычисляется только один раз). В зависимости от значения выполняется один из блоков `case`, в котором указано это значение. Если ни одно из значений не описано в блоках `case`, то выполняется блок `default` (если он указан).

Перепишем нашу программу определения операционной системы (см. листинг 5.14), заменив оператор `if...else` на `switch` (листинг 5.16).

Листинг 5.16. Использование оператора `switch`

```
<b>Какой операционной системой вы пользуетесь?</b><br><br>
<form action="<?=$_SERVER['SCRIPT_NAME']?>">
<select name="os">
<option value="0" selected>Не выбрано</option>
<option value="1">Windows XP</option>
<option value="2">Windows 7</option>
<option value="3">Windows 8</option>
<option value="4">Windows 10</option>
<option value="5">Другая</option>
</select>
<input type="submit" value="Выбрал">
</form>
<?php
if (isset($_GET['os'])) {
   switch($_GET['os']) {
      case '1':
         echo 'Вы выбрали — Windows XP'; break;
      case '2':
         echo 'Вы выбрали — Windows 7'; break;
      case '3':
         echo 'Вы выбрали — Windows 8'; break;
      case '4':
         echo 'Вы выбрали — Windows 10'; break;
      case '5':
         echo 'Вы выбрали — Другая'; break;
      case '0':
         echo 'Вы не выбрали операционную систему'; break;
      default:
         echo 'Мы не смогли определить вашу операционную систему';
   }
}
?>
```

Как видно из примера, в конце каждого блока `case` указан оператор `break`. Этот оператор позволяет досрочно выйти из оператора выбора `switch`. Зачем это нужно? Если не указать оператор `break` в конце блока `case`, то будет выполняться следующий блок `case` независимо от указанного значения. В большинстве случаев такая ситуация приводит к ошибкам, однако в некоторых случаях это может быть полезным. Например, можно выполнить одни и те же инструкции при разных значениях, разместив инструкции в конце диапазона значений:

```
$ch = 'c';
switch ($ch) {
   case 'a':
   case 'b':
   case 'c':
      echo "a, b или c";
      break;
   case 'd':
      echo "Только d";
}
```

Существует также следующий формат оператора выбора `switch`:

```
<?php switch (<Переменная или выражение>):
case <Значение 1>: ?>
   <Текст, выводимый, если условие истинно>
<?php break;
case <Значение 2>: ?>
   <Текст, выводимый, если условие истинно>
<?php break;
default: ?>
   <Текст, выводимый, если все условия ложны>
<?php endswitch; ?>
```

Этот формат удобно использовать для вывода HTML-кода:

```
<?php
$ch = 'a';
?>
<?php switch ($ch):
case 'a': ?>
Значение a
<?php break;
case 'b': ?>
Значение b
<?php break;
default: ?>
Другое значение
<?php endswitch; ?>
```

5.4. Циклы. Многократное выполнение блока кода

Предположим, нужно вывести все числа от `1` до `100` по одному на строке. Обычным способом пришлось бы писать 100 строк кода:

```
echo "1<br>\n";
echo "2<br>\n";
...
echo "100<br>\n";
```

С помощью циклов то же действие можно выполнить одной строкой кода:

```
for ($i = 1; $i < 101; $i++) echo $i . "<br>\n";
```

Иными словами, циклы позволяют выполнить одни и те же инструкции многократно.

5.4.1. Цикл *for*

Цикл `for` используется для выполнения инструкций определенное число раз. Формат оператора:

```
for (<Начальное значение>; <Условие>; <Приращение>) {
   <Инструкции>
}
```

Здесь присутствуют следующие конструкции:

- `<Начальное значение>` — присваивает переменной-счетчику начальное значение;
- `<Условие>` — содержит логическое выражение. Пока логическое выражение возвращает значение `true`, выполняются инструкции внутри цикла. Обратите внимание: логическое выражение вычисляется на каждой итерации цикла;
- `<Приращение>` — задает изменение переменной-счетчика при каждой итерации.

Последовательность работы цикла `for`:

1. Переменной-счетчику присваивается начальное значение.
2. Проверяется условие: если оно истинно, выполняются инструкции внутри цикла, в противном случае выполнение цикла завершается.
3. Переменная-счетчик изменяется на величину, указанную в `<Приращение>`.
4. Переход к п. 2.

Цикл выполняется до тех пор, пока `<Условие>` не вернет `false`. Если это не случится, цикл будет бесконечным.

`<Приращение>` может не только увеличивать значение переменной-счетчика, но и уменьшать. Выведем все числа от `100` до `1`:

```
for ($i = 100; $i > 0; $i--) echo $i . "<br>\n";
```

<Приращение> может изменять значение переменной-счетчика не только на единицу. Выведем все четные числа от 1 до 100:

```
for ($i = 2; $i < 101; $i += 2) echo $i . "<br>\n";
```

Следует заметить, что выражение, указанное в параметре <Условие>, вычисляется на каждой итерации. Рассмотрим вывод элементов массива:

```
$arr = array(1, 2, 3);
for ($i = 0; $i < count($arr); $i++) {
   if ($i == 0) {
      $arr[] = 4; // Добавляем новые элементы
      $arr[] = 5; // для доказательства
   }
   echo $arr[$i] . " ";
} // Выведет: 1 2 3 4 5
```

В этом примере мы указываем функцию count() в параметре <Условие>, а внутри цикла (чтобы доказать вычисление на каждой итерации) добавляем в массив новые элементы. В итоге мы получили все элементы массива, включая новые. Чтобы этого избежать, следует вычисление размера массива указать в первом параметре:

```
$arr = array(1, 2, 3);
for ($i = 0, $c = count($arr); $i < $c; $i++) {
   if ($i == 0) {
      $arr[] = 4; // Добавляем новые элементы
      $arr[] = 5; // для доказательства
   }
   echo $arr[$i] . " ";
} // Выведет: 1 2 3
```

Все параметры цикла for являются необязательными. Хотя параметры можно не указывать, точки с запятой обязательно должны быть. Если все параметры не указаны, то цикл будет бесконечным. Чтобы выйти из бесконечного цикла, следует использовать оператор break:

```
$i = 1;                    // <Начальное значение>
for ( ; ; ) {              // Бесконечный цикл
   if ($i <= 10) {         // <Условие>
      echo $i, "<br>\n";
      $i++;                // <Приращение>
   }
   else {
      break;               // Выходим из цикла
   }
}
```

Существует также следующий формат оператора for:

```
<?php for (<Начальное значение>; <Условие>; <Приращение>): ?>
<Текст, выводимый указанное количество раз>
<?php endfor; ?>
```

Пример:

```
<?php for ($i = 1; $i < 101; $i++): ?>
Строка <?=$i?><br>
<?php endfor; ?>
```

5.4.2. Цикл *while*

Выполнение инструкций в цикле `while` продолжается до тех пор, пока логическое выражение истинно. Формат оператора:

```
<Начальное значение>;
while (<Условие>) {
   <Инструкции>;
   <Приращение>;
}
```

Последовательность работы цикла `while`:

1. Переменной-счетчику присваивается начальное значение.
2. Проверяется условие: если оно истинно, выполняются инструкции внутри цикла, иначе выполнение цикла завершается.
3. Переменная-счетчик изменяется на величину, указанную в `<Приращение>`.
4. Переход к п. 2.

Выведем все числа от `1` до `100`, используя цикл `while`:

```
$i = 1;
while ($i < 101) {
   echo $i . "<br>\n";
   $i++;
}
```

> ***Внимание!***
>
> Если `<Приращение>` не указано, то цикл будет бесконечным.

В качестве параметра `<Приращение>` не обязательно должна быть арифметическая операция. Например, при работе с базами данных в качестве `<Приращение>` будет перемещение к следующей строке, а условием выхода из цикла — последняя строка в базе данных. В этом случае `<Начальное значение>` — получение первой строки базы данных.

Существует также следующий формат оператора `while`:

```
<?php <Начальное значение>;
while (<Условие>): ?>
<Текст, выводимый указанное количество раз>
<?php <Приращение>;
endwhile; ?>
```

Пример:

```
<?php $i = 1;
while ($i < 101): ?>
Строка <?=$i?><br>
<?php $i++;
endwhile; ?>
```

5.4.3. Цикл *do...while*

Инструкции в теле цикла `do...while` выполняются до тех пор, пока логическое выражение истинно. Но, в отличие от цикла `while`, условие проверяется не в начале цикла, а в конце. Поэтому инструкции внутри цикла `do...while` выполнятся минимум один раз.

Формат оператора `do...while`:

```
<Начальное значение>;
do {
   <Инструкции>;
   <Приращение>;
} while (<Условие>);
```

Последовательность работы цикла `do...while`:

1. Переменной-счетчику присваивается начальное значение.
2. Выполняются инструкции внутри цикла.
3. Переменная-счетчик изменяется на величину, указанную в `<Приращение>`.
4. Проверяется условие: если оно истинно, происходит переход к п. 2, а если нет — выполнение цикла завершается.

Выведем все числа от `1` до `100`, используя цикл `do...while`:

```
$i = 1;
do {
   echo $i . "<br>\n";
   $i++;
}
while ($i < 101);
```

> ***Внимание!***
> Если `<Приращение>` не указано, то цикл будет бесконечным.

5.4.4. Цикл *foreach*

Цикл `foreach` используется для перебора элементов массива:

```
$arr = array('Один', 'Два', 'Три', 'Четыре');
foreach ($arr as $value) {
   echo $value . "<br>\n";
}
```

Перебрать элементы ассоциативного массива можно следующим образом:

```
$arr['Один'] = 1;
$arr['Два'] = 2;
$arr['Три'] = 3;
$arr['Четыре'] = 4;
foreach ($arr as $key => $value) {
   echo $key . ' =&gt; ' . $value . "<br>\n";
}
```

Если параметр в цикле `foreach` не является массивом, интерпретатор выведет сообщение об ошибке:

```
$arr = '';
foreach ($arr as $key => $value) {
   echo $key . ' =&gt; ' . $value . "<br>\n";
}
// Ошибка: Warning: Invalid argument supplied for foreach()
```

По этой причине перед вызовом цикла `foreach` необходимо проверить тип переменной, например, с помощью функции `is_array()` или `is_iterable()`:

```
if (isset($arr) && is_array($arr)) {
   // Проверка существования и типа переменной
   foreach ($arr as $key => $value) {
      echo $key . ' =&gt; ' . $value . "<br>\n";
   }
}
```

С помощью оператора `foreach` можно перебирать двумерные массивы с распаковкой вложенного массива. В этом случае нужно указать переменные внутри оператора `list()`. Через эти переменные будут доступны текущие значения элементов массива:

```
$arr = [
   [1, 2],
   [3, 4]
];
foreach ($arr as list($a, $b)) {
   echo $a . ', ' . $b . "<br>\n";
}
```

Следует учитывать, что переменная, указанная в параметре оператора `foreach`, внутри тела цикла содержит лишь копию элемента массива. Если попытаться внутри цикла умножить все элементы массива на `2`, то ничего не получится:

```
$arr = [1, 2, 3];
foreach ($arr as $value) {
   $value *= 2;
}
print_r($arr); // Array ( [0] => 1 [1] => 2 [2] => 3 )
```

Чтобы изменить значения элементов массива внутри цикла `foreach`, нужно перед именем переменной указать символ `&`. В этом случае будет создаваться ссылка на элемент массива. После окончания выполнения цикла нужно обязательно разорвать ссылку с последним элементом массива с помощью оператора `unset()`, иначе можно получить неожиданный результат в дальнейшем. Вот пример умножения всех элементов массива на `2`:

```
$arr = [1, 2, 3];
foreach ($arr as &$value) {
   $value *= 2;
}
// Разрываем ссылку
unset($value);
print_r($arr); // Array ( [0] => 2 [1] => 4 [2] => 6 )
```

Существуют также следующие форматы оператора `foreach`:

```
<?php foreach (<Массив> as <Переменная>): ?>
<Текст, выводимый внутри цикла>
<?php endforeach; ?>
<?php foreach (<Массив> as <Ключ> => <Значение>): ?>
<Текст, выводимый внутри цикла>
<?php endforeach; ?>
```

Пример:

```
<?php
$arr = [1, 2, 3];
$arr2 = [];
$arr2['Один'] = 1;
$arr2['Два'] = 2;
$arr2['Три'] = 3;
?>
<?php foreach ($arr as $value): ?>
Значение <?=$value?><br>
<?php endforeach; ?>
<?php foreach ($arr2 as $key => $value): ?>
Ключ <?=$key?> значение <?=$value?><br>
<?php endforeach; ?>
```

5.4.5. Оператор *continue*. Переход на следующую итерацию цикла

Оператор `continue` позволяет перейти к следующей итерации цикла до завершения выполнения всех инструкций внутри цикла. Формат:

```
continue [<Уровень>];
```

Выведем все числа от 1 до 100, кроме чисел от 5 до 10 включительно:

```
for ($i = 1; $i < 101; $i++) {
   if ($i > 4 && $i < 11) continue;
   echo $i . "<br>\n";
}
```

5.4.6. Оператор *break*. Прерывание цикла

Оператор `break` позволяет прервать выполнение цикла досрочно. Формат:

```
break [<Уровень>];
```

Для примера выведем все числа от 1 до 100 еще одним способом:

```
for ($i = 1; ; $i++) {
   if ($i > 100) break;
   echo $i . "<br>\n";
}
```

Здесь мы оставили условие цикла пустым, и это значит, что цикл будет продолжаться бесконечно. Однако, благодаря наличию оператора `break`, выполнение цикла прерывается, как только будет напечатано 100 строк.

> **Внимание!**
>
> Оператор `break` прерывает выполнение цикла, а не программы, т. е. далее будет выполнена инструкция, следующая сразу за циклом.

5.4.7. Оператор *goto*

С помощью оператора безусловного перехода `goto` можно передать управление в какое-либо место программы. Оператор имеет следующий формат:

```
goto <Метка>;
```

Значение в параметре `<Метка>` должно быть допустимым идентификатором. Место в программе, в которое передается управление, помечается одноименной меткой, после которой указывается двоеточие. В качестве примера имитируем цикл и выведем числа от 1 до 100:

```
$i = 1;
BLOCK_START: {
   if ($i > 100) goto BLOCK_END;
   echo $i . "<br>\n";
   $i++;
   goto BLOCK_START;
}
BLOCK_END:;
```

Как видно из примера, фигурные скобки можно использовать не только применительно к условным операторам и циклам, но и как отдельную конструкцию. Фрагмент кода, заключенный в фигурные скобки, называется *блоком*.

> **ПРИМЕЧАНИЕ**
>
> Следует избегать использования оператора `goto`, т. к. его применение делает программу слишком запутанной и может привести к неожиданным результатам.

5.5. Числа

В языке PHP для хранения чисел предназначены следующие типы данных:

- `int` (или `integer`) — целые числа;
- `float` (или `double`) — вещественные числа.

Размер `int` зависит от разрядности операционной системы. При этом в Windows тип `int` всегда кодируется 32-мя битами. Получить текущий размер можно с помощью константы `PHP_INT_SIZE`:

```
var_dump(PHP_INT_SIZE);      // int(4)
```

Диапазон значений позволяют увидеть константы `PHP_INT_MIN` и `PHP_INT_MAX`:

```
var_dump(PHP_INT_MIN);       // int(-2147483648)
var_dump(PHP_INT_MAX);       // int(2147483647)
```

Если значение выходит за диапазон допустимых значений для типа `int`, то тип `int` автоматически преобразуется в тип `float`:

```
var_dump(PHP_INT_MAX + 1);  // float(2147483648)
```

Целочисленное значение задается в десятичной, двоичной, восьмеричной или шестнадцатеричной форме. Двоичные числа начинаются с комбинации символов `0b` (или `0B`) и могут содержать цифры `0` или `1`. Восьмеричные числа начинаются с нуля и содержат цифры от `0` до `7`. Шестнадцатеричные числа начинаются с комбинации символов `0x` (или `0X`) и могут содержать цифры от `0` до `9` и буквы от `A` до `F` (регистр букв не имеет значения). Двоичные, восьмеричные и шестнадцатеричные значения преобразуются в десятичное значение:

```
// Двоичное значение
var_dump(0b01110111);          // int(119)
// Восьмеричное значение
var_dump(0167);                // int(119)
// Шестнадцатеричное значение
var_dump(0x77);                // int(119)
var_dump(0xFF);                // int(255)
// Десятичное значение
var_dump(119);                 // int(119)
```

Вещественное число может содержать точку и (или) экспоненту, начинающуюся с буквы `E` (регистр не имеет значения):

```
var_dump(20.0);                // float(20)
var_dump(12.1e20);             // float(1.21E+21)
var_dump(.123);                // float(0.123)
var_dump(47.E-10);             // float(4.7E-9)
```

При выполнении операций над вещественными числами следует учитывать ограничения точности вычислений. Например, результат следующей инструкции может показаться странным:

```
var_dump(0.3 - 0.1 - 0.1 - 0.1); // float(-2.7755575615629E-17)
```

Ожидаемым был бы результат `0.0`, но, как видно из примера, мы получили совсем другой результат (`-2.7755575615629E-17`). Он очень близок к нулю, но не равен нулю. Учитывайте это при указании вещественных чисел в качестве значения счетчика внутри цикла. Попытка проверить значение на равенство может привести к бесконечному циклу.

В логическом контексте число `0` трактуется как `false`, тогда как любое другое число — как `true`:

```
var_dump( (bool)0 );     // bool(false)
var_dump( (bool)1 );     // bool(true)
var_dump( (bool)-1 );    // bool(true)
var_dump( (bool)INF );   // bool(true)
var_dump( (bool)NAN );   // bool(true)
```

Если выполнить приведение логических значений к типу `int`, то `false` будет приведено к `0`, а `true` — к `1`:

```
var_dump( (int)false ); // int(0)
var_dump( (int)true );  // int(1)
```

5.5.1. Математические константы

В языке PHP определены следующие стандартные константы:

- `M_PI` — содержит число π. Получить значение константы можно также с помощью функции `pi()`:

  ```
  var_dump( M_PI );  // float(3.1415926535898)
  var_dump( pi() );  // float(3.1415926535898)
  ```

- `M_E` — содержит значение константы `e`:

  ```
  var_dump( M_E );   // float(2.718281828459)
  ```

5.5.2. Основные функции для работы с числами

Приведем основные функции для работы с числами:

- `abs()` — возвращает абсолютное значение:

  ```
  var_dump( abs(-1) );      // int(1)
  ```

- `pow(<Число>, <Степень>)` — возведение `<Число>` в `<Степень>`:

  ```
  var_dump( pow(10, 2) );  // int(100)
  var_dump( 10 ** 2 );     // int(100)
  ```

- `sqrt()` — квадратный корень:

```
var_dump( sqrt(100) );   // float(10)
```

- `exp()` — экспонента;
- `log()` — натуральный логарифм;
- `log10()` — десятичный логарифм;
- `max(<Список чисел через запятую>)` — максимальное значение:

```
var_dump( max(10, 3) );  // int(10)
```

- `min(<Список чисел через запятую>)` — минимальное значение:

```
var_dump( min(10, 3) );  // int(3)
```

5.5.3. Округление чисел

Для округления чисел предназначены следующие функции:

- `ceil()` — возвращает значение, округленное до ближайшего большего значения:

```
var_dump( ceil(1.49) );   // float(2)
var_dump( ceil(1.50) );   // float(2)
var_dump( ceil(1.51) );   // float(2)
```

- `floor()` — возвращает значение, округленное до ближайшего меньшего значения:

```
var_dump( floor(1.49) );  // float(1)
var_dump( floor(1.50) );  // float(1)
var_dump( floor(1.51) );  // float(1)
```

- `round()` — возвращает число, округленное до ближайшего меньшего целого, — для чисел с дробной частью, меньшей `0.5`, или значение, округленное до ближайшего большего целого, — для чисел с дробной частью, большей или равной `0.5`:

```
var_dump( round(1.49) );  // float(1)
var_dump( round(1.50) );  // float(2)
var_dump( round(1.51) );  // float(2)
```

5.5.4. Тригонометрические функции

В языке PHP доступны следующие основные тригонометрические функции:

- `sin()`, `cos()`, `tan()` — стандартные тригонометрические функции (синус, косинус, тангенс). Значение указывается в радианах:

```
var_dump( sin(deg2rad(90)) );  // float(1)
```

- `asin()`, `acos()`, `atan()` — обратные тригонометрические функции (арксинус, арккосинус, арктангенс). Значение возвращается в радианах;

- `deg2rad()` — преобразует градусы в радианы:

  ```
  var_dump( deg2rad(180) );      // float(3.1415926535898)
  ```

- `rad2deg()` — преобразует радианы в градусы:

  ```
  var_dump( rad2deg(M_PI) );     // float(180)
  ```

5.5.5. Преобразование строки в число

Числа, вводимые пользователем, например, в форме, представлены в виде строки. Чтобы в дальнейшем использовать эти числа, необходимо выполнить преобразование строки в число. Для этого в языке PHP предназначены следующие функции:

- `intval()` — преобразует значение в целое число. Во втором параметре можно указать систему счисления. Если преобразовать не удалось, то функция вернет число `0`. Формат:

  ```
  intval(mixed $var[, int $base=10]) : int
  ```

  Пример:

  ```
  // Двоичное значение
  var_dump( intval("0b1110111", 2) );  // int(119)
  // Восьмеричное значение
  var_dump( intval("0167", 8) );       // int(119)
  // Шестнадцатеричное значение
  var_dump( intval("0x77", 16) );      // int(119)
  var_dump( intval("0xFF", 16)) ;      // int(255)
  // Десятичное значение
  var_dump( intval("119", 10) );       // int(119)
  var_dump( intval("9строка") );       // int(9)
  var_dump( intval("строка9") );       // int(0)
  var_dump( intval("строка") );        // int(0)
  ```

  Для десятичных значений достаточно выполнить приведение типов:

  ```
  var_dump( (int) "119" );             // int(119)
  var_dump( (int) "0119" );            // int(119)
  var_dump( (int) "9строка" );         // int(9)
  var_dump( (int) "строка9" );         // int(0)
  var_dump( (int) "строка" );          // int(0)
  ```

- `bindec()` — преобразует двоичное значение в десятичное число:

  ```
  var_dump( bindec("1110111") );       // int(119)
  ```

- `octdec()` — преобразует восьмеричное значение в десятичное число:

  ```
  var_dump( octdec("167") );           // int(119)
  ```

- `hexdec()` — преобразует шестнадцатеричное значение в десятичное число:

  ```
  var_dump( hexdec("77") );            // int(119)
  var_dump( hexdec("FF") );            // int(255)
  ```

- floatval() — преобразует значение в вещественное число. Если преобразовать не удалось, то функция вернет число 0:

```
var_dump( floatval("1,2") );          // float(1)
var_dump( floatval("1.2") );          // float(1.2)
var_dump( floatval("1.2строка") );    // float(1.2)
var_dump( floatval("строка1.2") );    // float(0)
```

Можно также выполнить приведение типов:

```
var_dump( (float) "1,2" );            // float(1)
var_dump( (float) "1.2" );            // float(1.2)
var_dump( (float) "1.2строка" );      // float(1.2)
var_dump( (float) "строка1.2" );      // float(0)
```

5.5.6. Преобразование числа в строку

Чтобы преобразовать число в строку, достаточно выполнить операцию конкатенации:

```
var_dump( 10 . " " . 1.2);           // string(6) "10 1.2"
```

Можно также воспользоваться следующими функциями:

- number_format() — позволяет преобразовать число в отформатированную строку. Имеет следующий синтаксис:

```
number_format(<Число>[, <Число знаков после запятой>[,
              <Десятичный разделитель>, <Разделитель тысяч>]])
```

Пример:

```
$x = 1234567.126;
var_dump( number_format($x) );     // string(9) "1,234,567"
var_dump( number_format($x, 2) ); // string(12) "1,234,567.13"
var_dump( number_format($x, 2, ',', ' ') );
// string(12) "1 234 567,13"
```

- sprintf() — позволяет преобразовать число в отформатированную строку. Формат функции:

```
sprintf(string $format, mixed $args=null, mixed $_=null) : string
```

В параметре $format указывается строка специального формата, внутри которой с помощью спецификаторов задаются правила форматирования. Какие спецификаторы используются, мы рассмотрим немного позже при изучении форматирования строк. В параметре $args через запятую указываются различные значения:

```
var_dump( sprintf("%d", 10) );        // string(2) "10"
var_dump( sprintf("%f", 1.126) );     // string(8) "1.126000"
var_dump( sprintf("%.2f", 1.126) );   // string(4) "1.13"
```

Формат десятичного разделителя зависит от настроек локали:

```
setlocale(LC_ALL, 'Russian_Russia'); // Настройка локали
var_dump( sprintf("%.2f", 1.126) );   // string(4) "1,13"
```

С помощью функции `sprintf()` можно также преобразовать десятичное число в двоичное, восьмеричное или шестнадцатеричное значение:

```
// Двоичное значение
var_dump( sprintf("%b", 119) );  // string(7) "1110111"
// Восьмеричное значение
var_dump( sprintf("%o", 119) );  // string(3) "167"
// Шестнадцатеричное значение
var_dump( sprintf("%x", 119) );  // string(2) "77"
var_dump( sprintf("%X", 255) );  // string(2) "FF"
```

- `base_convert()` — позволяет преобразовать число, записанное в одной системе счисления, в другую. Имеет следующий формат:

```
base_convert(<Содержащая число строка>,
     <Исходная система счисления>, <Нужная система счисления>)
```

Пример:

```
var_dump( base_convert(9, 10, 2) );     // string(4) "1001"
var_dump( base_convert("A", 16, 10) );  // string(2) "10"
```

- `decbin()` — преобразует десятичное число в двоичное:

```
var_dump( decbin(119) );                // string(7) "1110111"
```

- `decoct()` — преобразует десятичное число в восьмеричное;

```
var_dump( decoct(119) );                // string(3) "167"
```

- `dechex()` — преобразует десятичное число в шестнадцатеричное:

```
var_dump( dechex(119) );                // string(2) "77"
```

5.5.7. Генерация псевдослучайных чисел

Для генерации псевдослучайных чисел предназначены следующие функции:

- `mt_rand([<Начало диапазона>, <Конец диапазона>])` — генерирует случайное число от `<Начало диапазона>` до `<Конец диапазона>` включительно:

```
echo mt_rand(10, 100);
```

Если параметры не указаны, то возвращает значение от `0` до значения функции `mt_getrandmax()`:

```
echo mt_rand() . "\n";
echo mt_getrandmax(); // 2147483647
```

- `random_int(<Начало диапазона>, <Конец диапазона>)` — генерирует случайное целое число от `<Начало диапазона>` до `<Конец диапазона>` включительно:

```
echo random_int(10, 100);
```

- `mt_srand()` — настраивает генератор случайных чисел на новую последовательность. Формат функции:

```
mt_srand([int $seed[, int $mode = MT_RAND_MT19937]]) : void
```

Значением параметра `$seed` часто служит результат выполнения функции `time()`, которая возвращает число секунд, прошедшее с 1 января 1970 года. В параметре `$mode` можно указать константы `MT_RAND_MT19937` или `MT_RAND_PHP`. Пример:

```
mt_srand(time());
echo mt_rand(10, 100);
```

Для примера создадим генератор паролей произвольной длины (листинг 5.17). Для этого добавляем в массив `$arr` все разрешенные символы, а далее в цикле получаем содержимое массива по случайному индексу. По умолчанию будет выдаваться пароль из восьми символов.

Листинг 5.17. Генератор паролей

```
<?php
function passwGenerator(int $length = 8) : string {
   if ($length < 1) return '';
   $arr =
   array('a','b','c','d','e','f','g','h','i','j','k','l',
      'm','n','o','p','q','r','s','t','u','v','w','x','y','z',
      'A','B','C','D','E','F','G','H','I','J','K','L',
      'M','N','O','P','Q','R','S','T','U','V', 'W',
      'X','Y','Z','1','2','3','4','5','6','7','8','9','0');
   $password = '';
   $max = count($arr) - 1;
   for ($i = 0; $i < $length; $i++) {
      $password .= $arr[ mt_rand(0, $max) ];
   }
   return $password;
}
echo passwGenerator(10); // Выведет что-то вроде 7Jv1Wwoz3u
```

5.5.8. Бесконечность и значение *NaN*

Деление вещественного числа на `0.0` приведет к значению плюс или минус `INF` (бесконечность), а деление вещественного числа `0.0` на `0.0` — к значению `NaN` (нет числа). При этом выводятся предупреждающие сообщения о делении на `0`:

```
var_dump( 10.0 / 0 );  // float(INF)  + Warning: Division by zero
var_dump( -10.0 / 0 ); // float(-INF) + Warning: Division by zero
var_dump( 0.0 / 0 );   // float(NAN)  + Warning: Division by zero
```

Для проверки соответствия этим значениям следует воспользоваться следующими функциями:

- `is_infinite()` — возвращает `true`, если значение равно плюс или минус бесконечность, и `false` — в противном случае;
- `is_finite()` — возвращает `true`, если значение не равно плюс или минус бесконечность или значению `NaN`, и `false` — в противном случае;

- `is_nan()` — возвращает `true`, если значение равно `NaN`, и `false` — в противном случае:

```
@$x = 10.0 / 0;
@$y = 0.0 / 0;
$z = 5;
var_dump( is_infinite($x) );  // bool(true)
var_dump( is_infinite($y) );  // bool(false)
var_dump( is_infinite($z) );  // bool(false)

var_dump( is_nan($x) );       // bool(false)
var_dump( is_nan($y) );       // bool(true)
var_dump( is_nan($z) );       // bool(false)

var_dump( is_finite($x) );    // bool(false)
var_dump( is_finite($y) );    // bool(false)
var_dump( is_finite($z) );    // bool(true)
```

5.6. Массивы

Массив — это нумерованный или именованный набор переменных. Переменная в массиве называется *элементом* массива, а ее позиция в массиве задается *индексом*, или *ключом*. Нумерация элементов массива по умолчанию начинается с нуля, а не с единицы. Это следует помнить. Общее число элементов в массиве называется *размером* массива. Массивы, ключами которых являются числа, часто называют *списками*, а массивы, ключами которых являются строки, — *ассоциативными массивами*.

В языке PHP граница между списками и ассоциативными массивами является условной, т. к. в качестве ключа можно указать и целое число, и строку, причем в одном и том же массиве.

5.6.1. Инициализация массива

Массив инициализируют тремя способами:

- поэлементно:

```
$arr = array();
$arr[0] = 'Ноль';
$arr[1] = 'Один';
$arr[2] = 'Два';
$arr[3] = 'Три';
```

  Кроме того, можно не указывать индекс. PHP автоматически присвоит элементу индекс, на единицу больший последнего, т. е. добавит элемент в конец массива:

```
$arr = [];
$arr[] = 'Ноль';
```

```
$arr[] = 'Один';
$arr[] = 'Два';
$arr[] = 'Три';
```

- указав все элементы массива сразу с помощью оператора `array()`:

```
$arr = array('Ноль', 'Один', 'Два', 'Три');
$arr2 = array(
   0 => 'Ноль',
   1 => 'Один',
   2 => 'Два',
   3 => 'Три'
);
```

- указав все элементы массива сразу внутри квадратных скобок:

```
$arr = ['Ноль', 'Один', 'Два', 'Три'];
$arr2 = [
   0 => 'Ноль',
   1 => 'Один',
   2 => 'Два',
   3 => 'Три'
];
```

Индексы массивов в PHP на самом деле являются ключами ассоциативных массивов, поэтому возможна нумерация с любого значения (по умолчанию с нуля), а также прерывание порядка следования индексов внутри массива:

```
$arr = array(
   3 => 'Ноль',
   'Один',
   8 => 'Два',
   'Три'
);
print_r( $arr );
/*
Array
(
    [3] => Ноль
    [4] => Один
    [8] => Два
    [9] => Три
) */
```

С помощью функции `array_pad()` можно дополнить размер массива до указанной во втором параметре величины и присвоить новым элементам значение, заданное в третьем параметре. Создадим пять элементов со значениями `0`:

```
$arr = [];
$arr2 = array_pad($arr, 5, 0);
print_r($arr2);
// Array ( [0] => 0 [1] => 0 [2] => 0 [3] => 0 [4] => 0 )
```

5.6.2. Получение и изменение элемента массива

Обращение к элементам массива осуществляется с помощью квадратных или фигурных скобок, в которых указывается индекс элемента. Нумерация элементов массива по умолчанию начинается с нуля:

```
$arr = array('Ноль', 'Один', 'Два', 'Три');
$var = $arr[1];  // Переменной $var будет присвоено значение "Один"
$var2 = $arr{2}; // Переменной $var2 будет присвоено значение "Два"
```

Если элемент с указанным индексом не существует, то будет возвращено значение `null`. При этом выводится предупреждающее сообщение **Notice: Undefined offset**:

```
$arr = array('Ноль', 'Один', 'Два', 'Три');
$var = $arr[4];                               // Ошибка
$var = $arr[4] ?? "Значение по умолчанию"; // OK
```

Обратиться к элементам массива можно с помощью инструкции `list()`:

```
$arr = ['Ноль', 'Один', 'Два', 'Три'];
list($var1, $var2, $var3, $var4) = $arr;
echo $var2; // Переменной $var2 будет присвоено значение "Один"
```

Вместо инструкции `list()` допускается указание переменных через запятую внутри квадратных скобок:

```
$arr = ['Ноль', 'Один'];
[$var1, $var2] = $arr;
echo $var1 . ' ' . $var2; // Ноль Один
```

При желании можно добавить новый элемент массива или изменить значение существующего:

```
$arr[] = 'Четыре';
$arr[0] = 'Нуль';
```

5.6.3. Определение числа элементов массива

Получить число элементов массива позволяют функции `count()` и `sizeof()`:

```
$arr = array('Ноль', 'Один', 'Два');
echo count($arr);  // Выведет: 3
echo sizeof($arr); // Выведет: 3
```

Следует учитывать, что функция `count()` возвращает число существующих элементов массива. Если элемент не определен, то он не учитывается в подсчете. Об этой особенности важно помнить при переборе элементов с помощью цикла `for`. Например, в следующем примере элементы с индексами `0`, `1`, `2` и `5` не учитываются, т. к. нумерация начинается с индекса `3`, а элемента с индексом `5` нет:

```
$arr = array(3 => 'a', 'b', 6 => 'c');
echo count($arr);  // Выведет: 3
print_r( $arr );   // Array ( [3] => a [4] => b [6] => c )
```

5.6.4. Ассоциативные массивы

Основное отличие ассоциативных массивов от списков — возможность обращения к элементу массива не по числовому индексу, а по индексу, представляющему собой строку. Индексы ассоциативного массива называются *ключами*. Вот пример ассоциативного массива:

```
$arr = array();
$arr['Один'] = 1;
$arr['Два'] = 2;
$arr['Три'] = 3;
echo $arr['Один']; // 1
echo $arr{'Два'};  // 2
```

Кроме перечисления, для инициализации ассоциативных массивов используются инструкция `array()` и квадратные скобки:

```
$arr = array('Один' => 1, 'Два' => 2, 'Три' => 3);
echo $arr['Один']; // 1
$arr2 = [ 'Один' => 1, 'Два' => 2, 'Три' => 3 ];
echo $arr2['Три']; // 3
```

Для доступа к значению после имени переменной указываются квадратные или фигурные скобки, внутри которых задается ключ. Если указанный ключ не содержится в массиве, то будет возвращено значение `null` и выведено предупреждающее сообщение:

```
$arr = array('key1' => 1, 'key2' => 2, 'key3' => "2");
var_dump($arr['key1']); // int(1)
var_dump($arr['key5']); // NULL
// Notice: Undefined index: key5
```

Проверить существование элемента в массиве позволяет оператор `isset()`:

```
$arr = array('key1' => 1, 'key2' => 2, 'key3' => "2");
if (isset($arr['key5'])) $var = $arr['key5'];
else $var = 'Значение по умолчанию';
var_dump($var); // string(40) "Значение по умолчанию"
```

С помощью оператора `??` такую проверку можно упростить:

```
$arr = array('key1' => 1, 'key2' => 2, 'key3' => "2");
$var = $arr['key5'] ?? 'Значение по умолчанию';
var_dump($var); // string(40) "Значение по умолчанию"
```

Обратите внимание: если ключ существует, а элемент имеет значение `null`, то оператор `isset()` вернет значение `false`. Чтобы проверить наличие ключа, в этом случае следует использовать функцию `array_key_exists()`:

```
$arr = array('key1' => null);
var_dump( isset($arr['key1']) );             // bool(false)
var_dump( array_key_exists('key1', $arr) ); // bool(true)
```

Функции `array_keys()` и `array_values()` позволяют получить все ключи и все значения массива соответственно:

```
$arr = array('Один' => 1, 'Два' => 2, 'Три' => 3);
print_r( array_keys($arr) );
// Array ( [0] => Один [1] => Два [2] => Три )
print_r( array_values($arr) );
// Array ( [0] => 1 [1] => 2 [2] => 3 )
$arr2 = ['Ноль', 'Один', 'Два'];
print_r( array_keys($arr2) );
// Array ( [0] => 0 [1] => 1 [2] => 2 )
print_r( array_values($arr2) );
// Array ( [0] => Ноль [1] => Один [2] => Два )
```

Функция `array_keys()` имеет два дополнительных параметра:

```
array_keys(array $array[, mixed $search_value=null[,
           bool $strict=false]]) : array
```

Если в параметре `$search_value` указано значение, то будут возвращены только ключи, содержащие это значение. Обратите внимание: поиск выполняется по значениям массива, а не по именам ключей. Если в параметре `$strict` указано значение `true`, то при сравнении будет использоваться оператор строго равно (===). По умолчанию параметр `$strict` имеет значение `false`:

```
$arr = array('key 1' => 1, 'key 2' => 2, 'key 3' => "2");
print_r( array_keys($arr, 2, false) );
// Array ( [0] => key 2 [1] => key 3 )
print_r( array_keys($arr, 2, true) );
// Array ( [0] => key 2 )
```

5.6.5. Многомерные массивы

Любому элементу массива можно присвоить другой массив:

```
$arr = array();
$arr[0] = array(1, 2, 3, 4);
$arr[1] = [ 5, 6, 7, 8 ];
```

В этом случае получить значение массива можно, указав два индекса:

```
$var = $arr[0][2]; // Переменной $var будет присвоено значение 3
$var2 = $arr[1][0]; // Переменной $var2 будет присвоено значение 5
```

Создать многомерный ассоциативный массив можно так:

```
$arr = array();
$arr['Иванов'] = array('Имя' => 'Иван', 'Отчество' => 'Иванович',
                       'Год рождения' => 1966);
$arr['Семенов'] = [ 'Имя' => 'Сергей', 'Отчество' => 'Николаевич',
                    'Год рождения' => 1980 ];
```

Существует и другие способы:

```
$arr = array(
   'Иванов' => array('Имя' => 'Иван', 'Отчество' => 'Иванович',
                     'Год рождения' => 1966),
   'Семенов' => array('Имя' => 'Сергей', 'Отчество' => 'Николаевич',
                      'Год рождения' => 1980)
);
$arr2 = [
   'Иванов' => [ 'Имя' => 'Иван', 'Отчество' => 'Иванович',
                 'Год рождения' => 1966 ],
   'Семенов' => [ 'Имя' => 'Сергей', 'Отчество' => 'Николаевич',
                  'Год рождения' => 1980 ]
];
```

Доступ к элементу такого массива осуществляется путем указания двух ключей:

```
echo $arr['Иванов']['Год рождения']; // 1966
echo $arr2['Семенов']['Отчество'];   // Николаевич
```

Язык PHP допускает смешивание целочисленных индексов и ключей в виде строки в одном массиве. Однако на практике подобных смешиваний лучше избегать — выберите что-то одно: или список, или ассоциативный массив. Кроме того, количество элементов во вложенном массиве может быть произвольным, что позволяет создавать так называемые «зубчатые» многомерные массивы:

```
$arr = [
   'Иванов' => [ 'Имя' => 'Иван', 'Отчество' => 'Иванович',
                 'Год рождения' => 1966 ],
   2 => [ 'Фамилия' => 'Семенов', 'Имя' => 'Сергей',
          'Отчество' => 'Николаевич', 'Год рождения' => 1980 ]
];
echo $arr['Иванов']['Год рождения']; // 1966
echo $arr[2]['Фамилия'];             // Семенов
```

5.6.6. Создание копии массива

Для создания копии массива достаточно присвоить его другой переменной:

```
$arr = array(1, 2);
$arr[] = array(3, 4);
$arr2 = $arr;
$arr2[0] = 22;
$arr2[2][0] = 88;
print_r($arr);
// Array ( [0] => 1 [1] => 2 [2] => Array ( [0] => 3 [1] => 4 ) )
print_r($arr2);
// Array ( [0] => 22 [1] => 2 [2] => Array ( [0] => 88 [1] => 4 ) )
```

Если нужно, чтобы через другую переменную можно было изменять массив, то следует выполнить присваивание по ссылке:

```
$arr = array(1, 2);
$arr2 = &$arr; // Присваивание по ссылке
$arr2[0] = 22;
print_r($arr);  // Array ( [0] => 22 [1] => 2 )
print_r($arr2); // Array ( [0] => 22 [1] => 2 )
```

5.6.7. Слияние массивов

Для слияния двух ассоциативных массивов предусмотрен оператор +:

```
$arr1 = array('Один' => 1, 'Два' => 2);
$arr2 = array('Три' => 3, 'Четыре' => 4);
$arr3 = $arr1 + $arr2;
print_r( $arr3 );
// Array ( [Один] => 1 [Два] => 2 [Три] => 3 [Четыре] => 4 )
```

Для слияния двух списков оператор + не подходит. В этом случае используется функция `array_merge()`:

```
$arr1 = array('Один', 'Два');
$arr2 = array('Три', 'Четыре');
$arr3 = array_merge($arr1, $arr2);
print_r( $arr3 );
// Array ( [0] => Один [1] => Два [2] => Три [3] => Четыре )
```

> ***Внимание!***
> Если один из параметров в функции `array_merge()` не является массивом, интерпретатор выведет сообщение об ошибке.

Объединить два массива в ассоциативный массив позволяет функция `array_combine()`. Элементы массива, указанного в параметре `$keys`, станут ключами, а элементы массива, указанного в параметре `$values`, — значениями. Формат функции:

```
array_combine(array $keys, array $values) : array
```

Пример:

```
$keys = array('Один', 'Два');
$values = array(1, 2);
$arr = array_combine($keys, $values);
print_r( $arr );
// Array ( [Один] => 1 [Два] => 2 )
```

Функция `array_fill_keys()` позволяет создать ассоциативный массив с ключами из списка и присвоить всем элементам указанное значение:

```
$keys = array('Один', 'Два');
$arr = array_fill_keys($keys, 0);
print_r( $arr );
// Array ( [Один] => 0 [Два] => 0 )
```

5.6.8. Перебор элементов массива

Для перебора массивов лучше всего подходит цикл `foreach`, хотя можно воспользоваться и циклом `for`, и циклом `while`.

Цикл *foreach*

Цикл `foreach` позволяет работать как со списками:

```
$arr = array(1, 2, 3, 4);
foreach ($arr as $value) {
   echo $value . ' ';
} // 1 2 3 4
echo "<br>\n";
foreach ($arr as $index => $value) {
   echo $index . '=&gt;' . $value . ' ';
} // 0=>1 1=>2 2=>3 3=>4
```

так и с ассоциативными массивами:

```
$arr = array('Один' => 1, 'Два' => 2, 'Три' => 3);
foreach ($arr as $value) {
   echo $value . ' ';
} // 1 2 3
echo "<br>\n";
foreach ($arr as $key => $value) {
   echo $key . '=&gt;' . $value . ' ';
} // Один=>1 Два=>2 Три=>3
```

Для перебора многомерного массива один цикл `foreach` вкладывают в другой:

```
$arr = [
   [ 1, 2, 3, 4 ],
   [ 5, 6, 7, 8 ]
];
foreach ($arr as $v) {
   foreach ($v as $i) {
      echo $i . " ";
   }
   echo "; ";
} // 1 2 3 4 ; 5 6 7 8 ;
```

С помощью оператора `foreach` можно перебирать двумерные массивы с распаковкой вложенного массива. В этом случае нужно указать переменные внутри оператора `list()` или внутри квадратных скобок. Через эти переменные будут доступны текущие значения элементов массива:

```
$arr = [
   [ 1, 2 ],
   [ 3, 4 ]
];
```

```
foreach ($arr as list($a, $b)) {
// foreach ($arr as [$a, $b]) {
   echo $a . ',' . $b . "; ";
} // 1,2; 3,4;
```

Следует учитывать, что переменная, указанная в параметре оператора `foreach`, содержит лишь копию элемента массива. Если попытаться внутри цикла умножить все элементы массива на `2`, то ничего не получится:

```
$arr = [1, 2, 3];
foreach ($arr as $value) {
   $value *= 2;
}
print_r($arr); // Array ( [0] => 1 [1] => 2 [2] => 3 )
```

Чтобы изменить значения элементов массива внутри цикла `foreach`, нужно перед именем переменной указать символ `&`. В этом случае будет создаваться ссылка на элемент массива. После окончания выполнения цикла надо обязательно разорвать ссылку с последним элементом массива с помощью оператора `unset()`, иначе можно получить неожиданный результат в дальнейшем. Вот пример умножения всех элементов массива на `2`:

```
$arr = [1, 2, 3];
foreach ($arr as &$value) {
   $value *= 2;
}
// Разрываем ссылку
unset($value);
print_r($arr); // Array ( [0] => 2 [1] => 4 [2] => 6 )
```

Можно также поступить следующим образом:

```
$arr = [1, 2, 3];
foreach ($arr as $index => $value) {
   $arr[$index] *= 2;
}
print_r($arr); // Array ( [0] => 2 [1] => 4 [2] => 6 )
```

Цикл *for*

Цикл `for` используется, например, так:

```
$arr = [1, 2, 3];
for ($i = 0, $c = count($arr); $i < $c; $i++) {
   echo $arr[$i] . ' ';
} // 1 2 3
```

Следует с осторожностью пользоваться циклом `for`, т. к. функция `count()` возвращает число существующих элементов массива. Если элемент не определен, то он не учитывается в подсчете. Например, следующий код выведет не все элементы массива:

```
<?php
// Отключаем вывод предупреждающих сообщений
error_reporting(E_ALL & ~E_NOTICE);
$arr = [1 => 1, 2, 3]; // Нумерация с единицы
echo count($arr); // 3
echo "<br>\n";
for ($i = 0, $c = count($arr); $i < $c; $i++) {
   echo $arr[$i] . '; ';
} // ; 1; 2;
```

Как видно из примера, мы не получили значение элемента с индексом `3`, зато попытались получить значение несуществующего элемента с индексом `0`. Если воспользуемся в этом случае циклом `foreach`, то проблем не возникнет:

```
$arr = [1 => 1, 2, 3]; // Нумерация с единицы
foreach ($arr as $value) {
   echo $value . '; ';
} // 1; 2; 3;
```

Для перебора ассоциативного массива применяются следующие конструкции:

```
$arr = [ 'Один' => 1, 'Два' => 2, 'Три' => 3 ];
for (reset($arr); ($key = key($arr)) !== null; next($arr)) {
   echo $key . '=&gt;' . $arr[$key] . ' ';
} // Один=>1 Два=>2 Три=>3
```

В этом случае мы воспользовались следующими функциями:

- `reset()` — устанавливает указатель на первый элемент массива и возвращает его значение или значение `false`, если массив пуст;
- `next()` — перемещает указатель на один элемент вперед и возвращает значение элемента или значение `false`, если достигнут конец массива;
- `key()` — возвращает ключ текущего элемента массива или значение `null`.

Для перебора элементов ассоциативного массива в обратном порядке предназначены другие функции:

- `end()` — устанавливает указатель на последний элемент массива и возвращает его значение или значение `false`, если массив пуст;
- `prev()` — перемещает указатель на один элемент массива назад и возвращает значение элемента или значение `false`, если достигнуто начало массива.

Кроме того, для получения текущего значения элемента массива можно использовать функцию `current()`. Если массив пустой, функция возвращает значение `false`:

```
$arr = [ 'Один' => 1, 'Два' => 2, 'Три' => 3 ];
for (end($arr); ($key = key($arr)) !== null; prev($arr)) {
   echo $key . '=&gt;' . current($arr) . ' ';
} // Три=>3 Два=>2 Один=>1
```

Цикл *while*

Цикл `while` также подходит для работы с массивами:

```
$arr = [ 'Один' => 1, 'Два' => 2, 'Три' => 3 ];
reset($arr);
while ( ($key = key($arr)) !== null ) {
   echo $key . '=&gt;' . current($arr) . ' ';
   next($arr);
} // Один=>1 Два=>2 Три=>3
```

Перебор элементов массива без использования циклов

До сих пор мы выводили содержимое массивов с помощью циклов. Того же эффекта можно достичь при использовании функции `array_walk()`. Она позволяет последовательно применять созданную нами функцию ко всем элементам массива. Формат функции:

```
array_walk(array &$array, callable $callback[,
           mixed $userdata=null]) : bool
```

В первом параметре указывается ссылка на массив, а во втором — название функции обратного вызова в виде строки (без круглых скобок) или анонимная функция. Через первый параметр функции обратного вызова будет доступно значение элемента массива, через второй — ключ, а через третий — пользовательские данные, которые переданы в параметре `$userdata`.

Например, вывод всех элементов массива будет выглядеть так:

```
<?php
function print_array($value, $key) {
   echo $key . '=&gt;' . $value . ' ';
}

$arr = [ 'Один' => 1, 'Два' => 2, 'Три' => 3 ];
array_walk($arr, 'print_array');
// Один=>1 Два=>2 Три=>3
// С использованием анонимной функции
array_walk($arr, function ($value, $key, $userdata) {
   echo $key . '=&gt;' . $value . $userdata;
}, '; ');
// Один=>1; Два=>2; Три=>3;
```

Чтобы иметь возможность изменить текущее значение элемента массива, необходимо передать в функцию ссылку на него. Это делается путем указания символа `&` перед именем переменной в описании функции. Можно также обратиться к массиву напрямую, предварительно указав инструкцию `global`. Например, прибавим число `10` двумя способами:

```
<?php
function change_array(&$value, $key, $var) {
```

```
   // Передача значения по ссылке
   $value += $var;
}

$arr = [ 'Один' => 1, 'Два' => 2, 'Три' => 3 ];
// Изменение значений
array_walk($arr, 'change_array', 10);
array_walk($arr, function ($value, $key, $var) {
   // Доступ к массиву внутри функции
   global $arr;
   $arr[$key] += $var;
}, 10);
print_r($arr); // Array ( [Один] => 21 [Два] => 22 [Три] => 23 )
```

Переменная `$var` получит значение, указанное в третьем параметре функции `array_walk()`.

5.6.9. Добавление и удаление элементов массива

Для добавления и удаления элементов массива предусмотрены следующие функции:

- `array_unshift(<Массив>, <Элементы>)` — добавляет элементы в начало массива:

```
$arr = [ 3, 4 ];
array_unshift($arr, 1, 2);
print_r($arr); // Array ( [0] => 1 [1] => 2 [2] => 3 [3] => 4 )
```

- конструкция `<Массив>[]` — добавляет элементы в конец массива:

```
$arr = [ 1, 2 ];
$arr[] = 3;
$arr[] = 4;
print_r($arr); // Array ( [0] => 1 [1] => 2 [2] => 3 [3] => 4 )
```

- `array_push(<Массив>, <Элементы>)` — добавляет элементы в конец массива:

```
$arr = [ 1, 2 ];
array_push($arr, 3, 4);
print_r($arr); // Array ( [0] => 1 [1] => 2 [2] => 3 [3] => 4 )
```

- `array_shift(<Массив>)` — удаляет первый элемент массива и возвращает его (или значение `null`, если элементов больше нет):

```
$arr = [ 1, 2, 3, 4 ];
print_r( array_shift($arr) ); // 1
print_r($arr); // Array ( [0] => 2 [1] => 3 [2] => 4 )
```

- `array_pop(<Массив>)` — удаляет последний элемент массива и возвращает его (или значение `null`, если элементов больше нет):

```
$arr = [ 1, 2, 3, 4 ];
print_r( array_pop($arr) ); // 4
print_r($arr); // Array ( [0] => 1 [1] => 2 [2] => 3 )
```

Удалить произвольный элемент массива позволяет оператор `unset()`. Обратите внимание: в этом случае оставшиеся элементы списка не изменяют своих позиций. Чтобы переиндексировать список, нужно вызвать функцию `array_values()`:

```
$arr = [ 1, 2, 3, 4 ];
unset($arr[1]);
print_r($arr); // Array ( [0] => 1 [2] => 3 [3] => 4 )
$arr = array_values($arr);
print_r($arr); // Array ( [0] => 1 [1] => 3 [2] => 4 )
```

> ***Примечание***
>
> С помощью рассмотренных в этом разделе функций можно реализовать такие структуры данных, как очередь и стек (очередь, в которой последний пришедший уходит первым).

5.6.10. Переворачивание и перемешивание массива

Функция `array_reverse()` возвращает массив, элементы которого следуют в обратном порядке относительно исходного массива:

```
$arr = array('Один', 'Два', 'Три', 'Четыре');
$arr = array_reverse($arr);
print_r($arr);
// Array ( [0] => Четыре [1] => Три [2] => Два [3] => Один )
```

Функция `shuffle()` «перемешивает» массив — элементы массива будут расположены в случайном порядке:

```
$arr = array('Один', 'Два', 'Три', 'Четыре');
shuffle($arr);
print_r($arr);
// Array ( [0] => Два [1] => Один [2] => Три [3] => Четыре )
```

5.6.11. Сортировка массива

Функция `sort()` позволяет отсортировать список в алфавитном порядке, а функция `rsort()` — в обратном:

```
$arr = array('Один', 'Два', 'Три', 'Четыре');
sort($arr);
print_r($arr);
// Array ( [0] => Два [1] => Один [2] => Три [3] => Четыре )
rsort($arr);
print_r($arr);
// Array ( [0] => Четыре [1] => Три [2] => Один [3] => Два )
```

Для сортировки ассоциативных массивов эти функции не применяются, т. к. они разрывают связь ключа со значением. Отсортировать ассоциативный массив можно или по ключам, или по значениям.

Для этого используются следующие функции:

- `asort()` — сортировка по значениям в алфавитном порядке;
- `arsort()` — сортировка по значениям в обратном порядке:

```
$arr = array('Один' => 1, 'Два' => 2, 'Три' => 3);
arsort($arr, SORT_NUMERIC);
print_r($arr);
// Array ( [Три] => 3 [Два] => 2 [Один] => 1 )
```

- `ksort()` — сортировка по ключам в алфавитном порядке;
- `krsort()` — сортировка по ключам в обратном порядке:

```
$arr = array('Один' => 1, 'Два' => 2, 'Три' => 3);
krsort($arr, SORT_STRING);
print_r($arr);
// Array ( [Три] => 3 [Один] => 1 [Два] => 2 )
```

Во втором параметре в функциях `sort()`, `asort()`, `arsort()`, `ksort()` и `krsort()` можно указать константы: `SORT_REGULAR` (типы не меняются), `SORT_NUMERIC` (сравнение чисел), `SORT_STRING` (сравнение строк), `SORT_LOCALE_STRING` (сравнение строк с учетом локали) и `SORT_NATURAL` (сравнение строк с использованием естественного упорядочения). Совместно с константами `SORT_STRING` и `SORT_NATURAL` можно указать константу `SORT_FLAG_CASE`. В этом случае сортировка будет выполняться без учета регистра символов (с кодировкой UTF-8 это не работает). Обратите внимание: чтобы сортировка была без учета регистра символов, для кодировки UTF-8 нужно указать константу `SORT_LOCALE_STRING` и настроить локаль:

```
$arr = array('единица1', 'Единый', 'Единица2');
sort($arr);
print_r($arr);
// Array ( [0] => Единица2 [1] => Единый [2] => единица1 )
setlocale(LC_ALL, 'ru_RU', 'Russian_Russia');
sort($arr, SORT_LOCALE_STRING);
print_r($arr);
// Array ( [0] => единица1 [1] => Единица2 [2] => Единый )
```

Отсортировать массив, используя естественное упорядочение, позволяют функции `natsort()` и `natcasesort()`:

```
$arr = array('val1', 'val2', 'val10');
sort($arr);     // Обычная сортировка
print_r($arr);
// Array ( [0] => val1 [1] => val10 [2] => val2 )
natsort($arr);  // Естественное упорядочение
print_r($arr);
// Array ( [0] => val1 [2] => val2 [1] => val10 )
```

Создание пользовательской сортировки

Если нужно изменить порядок стандартной сортировки, можно задать свою сортировку с помощью следующих функций:

- `usort()` — для пользовательской сортировки списков;
- `uksort()` — для пользовательской сортировки ассоциативных массивов по ключам;
- `uasort()` — для пользовательской сортировки ассоциативных массивов по значениям.

В качестве первого аргумента этим функциям передается массив, а второй аргумент должен содержать имя функции, сравнивающей два элемента. Функция сравнения принимает два параметра и должна возвращать:

- `положительное число` — если первый больше второго;
- `отрицательное число` — если второй больше первого;
- `0` — если элементы равны.

Например, стандартная сортировка по умолчанию зависит от регистра символов:

```
$arr = array('единица1', 'Единый', 'Единица2');
sort($arr);
print_r($arr);
// Array ( [0] => Единица2 [1] => Единый [2] => единица1 )
```

В результате мы получим неправильную сортировку, ведь `Единый` и `Единица2` больше `единица1`. Изменим стандартную сортировку на свою собственную, не учитывающую регистр (листинг 5.18).

Листинг 5.18. Сортировка без учета регистра

```
function cmp($str1, $str2) {
   // Преобразуем к нижнему регистру
   $str1 = mb_strtolower($str1, 'UTF-8');
   $str2 = mb_strtolower($str2, 'UTF-8');
   // Сравниваем
   return strcmp($str1, $str2);
}

$arr = array('единица1', 'Единый', 'Единица2');
usort($arr, "cmp");
print_r($arr);
// Array ( [0] => единица1 [1] => Единица2 [2] => Единый )
```

Для получения правильной сортировки мы приводим две переменные к одному регистру, а затем осуществляем сравнение с помощью функции `strcmp()`. Заметьте, что регистр самих элементов массива не изменяется, т. к. в функцию передаются копии значений.

5.6.12. Получение части массива

Получить часть массива позволяет функция `array_slice()`, имеющая следующий формат вызова:

```
array_slice(array $array, int $offset[, int $length=null[,
           bool $preserve_keys=false]]) : array
```

Функции передаются следующие параметры:

- `$array` — исходный массив;
- `$offset` — число элементов от начала (положительное значение) или от конца (отрицательное значение) массива, которое нужно пропустить;
- `$length` — число элементов, которое нужно получить из исходного массива. Если параметр опущен или имеет значение `null`, то элементы выбираются до конца массива. Если указано отрицательное значение, то оно задает смещение от конца массива;
- `$preserve_keys` — если параметр имеет значение `true`, то ключи массива сбрасываться не будут.

Пример:

```
$arr = array(1, 2, 3, 4, 5);
$arr2 = array_slice($arr, 2, 3);
print_r($arr2); // Array ( [0] => 3 [1] => 4 [2] => 5 )
$arr2 = array_slice($arr, -4, 3);
print_r($arr2); // Array ( [0] => 2 [1] => 3 [2] => 4 )
$arr2 = array_slice($arr, 2, null, false);
print_r($arr2); // Array ( [0] => 3 [1] => 4 [2] => 5 )
$arr2 = array_slice($arr, 2, -1, true);
print_r($arr2); // Array ( [2] => 3 [3] => 4 )
```

Полученную часть массива можно заменить одним элементом или массивом элементов с помощью функции `array_splice()`. Вызов функции осуществляется так:

```
array_splice(array &$input, int $offset[,
            int $length=count($input)[,
            mixed $replacement=array()]]) : array
```

Первые три параметра имеют такое же назначение, как и у функции `array_slice()`. Если параметр `$length` имеет значение `0`, то параметр `$offset` задает место вставки элементов. Четвертый параметр `$replacement` — один элемент или массив элементов, добавляемый вместо выбранных элементов. Функция возвращает массив с удаленными элементами:

```
$arr1 = array(1, 2, 3, 4, 5);
$arr2 = array(6, 7, 8);
array_splice($arr1, 2, 3, $arr2);
print_r($arr1);
```

```
// Array ( [0] => 1 [1] => 2 [2] => 6 [3] => 7 [4] => 8 )
array_splice($arr2, 2, 0, 88);
print_r($arr2);
// Array ( [0] => 6 [1] => 7 [2] => 88 [3] => 8 )
```

5.6.13. Преобразование переменных в массив

Функция `compact()` позволяет преобразовать переменные в ассоциативный массив. Ключами становятся имена переменных, а значениями — значения переменных:

```
$var1 = 1;
$var2 = 2;
$var3 = 3;
$arr = compact('var1', 'var2', 'var3');
print_r($arr);
// Array ( [var1] => 1 [var2] => 2 [var3] => 3 )
```

5.6.14. Преобразование массива в переменные

Функция `extract()` создает переменные с именами, соответствующими именам ключей, и значениями, соответствующими значениям элемента ассоциативного массива. Формат вызова функции:

```
extract(<Массив>[, <Способ>[, <Префикс>]])
```

Можно указывать следующие параметры:

- `<Массив>` — исходный ассоциативный массив;
- `<Способ>` — способ обработки конфликтных ситуаций. Может принимать следующие значения:
  - `EXTR_OVERWRITE` — если переменная существует, то ее значение перезаписывается (значение по умолчанию);
  - `EXTR_SKIP` — если переменная существует, то элемент массива пропускается;
  - `EXTR_PREFIX_SAME` — если переменная существует, то перед именем переменной будет добавлен префикс, указанный в параметре `<Префикс>`;
  - `EXTR_PREFIX_ALL` — перед именем всех переменных будет добавлен префикс, указанный в параметре `<Префикс>`;
  - `EXTR_PREFIX_INVALID` — если имя переменной некорректно, то будет добавлен префикс, указанный в параметре `<Префикс>`;
  - `EXTR_IF_EXISTS` — извлекает значения только тех переменных, которые уже существуют;
  - `EXTR_PREFIX_IF_EXISTS` — извлекает значения только тех переменных, которые уже существуют, при этом создавая новое имя с префиксом, указанным в параметре `<Префикс>`;
  - `EXTR_REFS` — извлекает переменные как ссылки.

Пример:

```
$var1 = 'Привет';
$arr = array('var1' => 'v1', 'var2' => 'v2', 'var3' => 'v3');
extract($arr, EXTR_PREFIX_SAME, 's');
echo "$var1 $s_var1 $var2 $var3";
// Привет v1 v2 v3
```

Так как переменная `$var1` существует, то перед именем создаваемой переменной будет добавлен префикс `s_`. Все остальные ключи будут преобразованы в одноименные переменные.

5.6.15. Заполнение массива значениями

Создать массив, содержащий диапазон чисел, можно либо с помощью цикла, либо с помощью функции `range()`. Функция имеет следующий формат:

```
range(mixed $start, mixed $end[, number $step=1]) : array
```

Создадим массив, состоящий из диапазона чисел от `1` до `10`:

```
$arr = range(1, 10);
foreach ($arr as $value) {
   echo $value . " ";
} // 1 2 3 4 5 6 7 8 9 10
```

В параметре `$step` можно указать шаг (значение по умолчанию: `1`):

```
$arr = range(1, 10, 2);
foreach ($arr as $value) {
   echo $value . " ";
} // 1 3 5 7 9
```

Если нужно заполнить массив одинаковым значением, то следует воспользоваться функцией `array_fill()`. Формат функции:

```
array_fill(int $start_index, int $num, mixed $value) : array
```

В первом параметре указывается начальный индекс, во втором — число элементов, а в третьем — значение. Заполним массив из `10` элементов числами `0`:

```
$arr = array_fill(0, 10, 0);
foreach ($arr as $value) {
   echo $value . " ";
} // 0 0 0 0 0 0 0 0 0 0
```

5.6.16. Преобразование массива в строку

Преобразовать массив в строку можно с помощью нескольких функций:

- `implode()` — преобразует массив в строку. Элементы добавляются через указанный разделитель:

  ```
  $arr = array(1, 2, 3, 4, 5);
  $str = implode(', ', $arr);
  ```

```
echo $str; // 1, 2, 3, 4, 5
$str = implode(' - ', $arr);
echo $str; // 1 - 2 - 3 - 4 - 5
```

- `join()` — полностью аналогична функции `implode()`;
- `serialize()` — позволяет преобразовать любой массив в строку специального формата:

```
$arr = array('Фамилия', 'Имя');
$str = serialize($arr);
echo $str; // a:2:{i:0;s:14:"Фамилия";i:1;s:6:"Имя";}
```

- `unserialize()` — используется для восстановления массива из строки, преобразованной с помощью функции `serialize()`:

```
$arr = array('Фамилия', 'Имя');
$str = serialize($arr);
$arr2 = unserialize($str);
print_r($arr2); // Array ( [0] => Фамилия [1] => Имя )
```

- `print_r()` — позволяет вывести структуру массива:

```
$arr = array('Один', 'Два', 'Три');
print_r($arr);
/*
Array
(
    [0] => Один
    [1] => Два
    [2] => Три
) */
```

- `var_dump()` — выводит подробную информацию о структуре массива:

```
$arr = array('Один', 2, 'Три');
var_dump($arr);
/*
array(3) {
  [0]=>
  string(8) "Один"
  [1]=>
  int(2)
  [2]=>
  string(6) "Три"
} */
```

Внимание!

Функции `print_r()` и `var_dump()` позволяют выводить не только структуру массивов, но и значения других переменных. Поэтому они часто применяются на этапе отладки программы.

5.6.17. Проверка наличия значения в массиве

Функция `in_array()` позволяет проверить наличие значения в массиве. Возвращает `true`, если значение присутствует. Формат функции:

```
in_array(mixed $needle, array $haystack[, bool $strict=false]) : bool
```

Параметр `$needle` может быть числом, строкой или массивом. Если указана строка, то сравнение проводится с учетом регистра символов. Если необязательный параметр `$strict` имеет значение `true` (значение по умолчанию: `false`), то дополнительно выполняется проверка соответствия типов данных.

Пример:

```
$arr = array('один', '1', 20);
var_dump(in_array('один', $arr));     // bool(true)
var_dump(in_array('Один', $arr));
// Выведет bool(false), т. к. не совпадает регистр символов
var_dump(in_array('1', $arr));        // bool(true)
var_dump(in_array(20, $arr, true));   // bool(true)
var_dump(in_array('20', $arr, true));
// Выведет bool(false), т. к. не совпадают типы данных
```

Выполнить поиск и получить индекс (ключ) первого найденного элемента с указанным значением позволяет функция `array_search()`. Если элемент не найден, функция возвращает значение `false`. Формат функции:

```
array_search(mixed $needle, array $haystack[,
             bool $strict=false]) : mixed
```

Пример:

```
$arr = array('один', '1', 20);
var_dump(array_search('один', $arr));       // int(0)
var_dump(array_search('20', $arr));         // int(2)
var_dump(array_search('20', $arr, true));
// Выведет bool(false), т. к. не совпадают типы данных
```

5.6.18. Операции со множествами

Множество — это набор уникальных элементов, с которым можно сравнивать другие элементы, чтобы определить, принадлежат ли они этому множеству. В языке PHP множеств нет, поэтому попробуем создать их самостоятельно на основе списков и ассоциативных массивов.

Чтобы оставить в массиве только уникальные элементы, следует воспользоваться функцией `array_unique()`. Значения сравниваются как строки. Ключи сохраняются. Формат функции:

```
array_unique(array $array[, int $sort_flags=SORT_STRING]) : array
```

Во втором параметре можно указать способ сортировки — константы: `SORT_REGULAR` (типы не меняются), `SORT_NUMERIC` (сравнение чисел), `SORT_STRING` (сравнение строк) и `SORT_LOCALE_STRING` (сравнение строк с учетом локали):

```
$arr = [ 1, 1, 1, 2, 2, 2, 3, 3, 3, 3 ];
$arr2 = array_unique($arr);
print_r($arr2); // Array ( [0] => 1 [3] => 2 [6] => 3 )
$arr2 = array_values($arr2);  // Переиндексация
print_r($arr2); // Array ( [0] => 1 [1] => 2 [2] => 3 )
$arr = array('Один' => 1, 'Два' => 2,
             'Три' => '1', 'Четыре' => 4);
$arr2 = array_unique($arr);
print_r($arr2);
// Array ( [Один] => 1 [Два] => 2 [Четыре] => 4 )
```

Рассмотрим основные операции со множествами на основе списков:

- объединение множеств (функция `array_merge()`):

```
$arr1 = array(1, 2, 3);
$arr2 = array(3, 4, 5);
$arr3 = array_merge($arr1, $arr2);
$arr3 = array_unique($arr3);  // Выбор уникальных элементов
$arr3 = array_values($arr3);  // Переиндексация
print_r($arr3);
// Array ( [0] => 1 [1] => 2 [2] => 3 [3] => 4 [4] => 5 )
```

- разница множеств (функция `array_diff()` или `array_udiff()`):

```
$arr1 = array(1, 2, 3);
$arr2 = array(1, 2, 4);
$arr3 = array_diff($arr1, $arr2);
$arr3 = array_unique($arr3);  // Выбор уникальных элементов
$arr3 = array_values($arr3);  // Переиндексация
print_r($arr3);               // Array ( [0] => 3 )
```

- пересечение множеств (функция `array_intersect()` или `array_uintersect()`):

```
$arr1 = array(3, 1, 2, 2, 3);
$arr2 = array(1, 2, 4);
$arr3 = array_intersect($arr1, $arr2);
$arr3 = array_unique($arr3);  // Выбор уникальных элементов
$arr3 = array_values($arr3);  // Переиндексация
print_r($arr3);               // Array ( [0] => 1 [1] => 2 )
```

- проверка наличия элемента (функция `in_array()`):

```
$arr = array(3, 1, 2, 2, 3);
$arr = array_unique($arr);    // Выбор уникальных элементов
$arr = array_values($arr);    // Переиндексация
var_dump(in_array(1, $arr));  // bool(true)
var_dump(in_array(5, $arr));  // bool(false)
```

В качестве элементов множества лучше использовать ключи ассоциативных массивов. Ключи всегда уникальны, и поиск ключа выполняется очень быстро. Однако тут есть ограничение — в качестве типа данных можно использовать или целое

число, или строку, другие типы данные не поддерживаются. В этом случае для объединения множеств следует использовать оператор +, для получения разницы — функцию `array_diff_key()` (или `array_diff_ukey()`), для получения пересечения — функцию `array_intersect_key()` (или `array_intersect_ukey()`), а для проверки существования элемента — оператор `isset()`:

```
$arr1 = array(1 => 1, 2 => 1, 3 => 1);
$arr2 = array(1 => 1, 2 => 1, 4 => 1);
// Объединение множеств
$arr3 = $arr1 + $arr2;
print_r($arr3); // Array ( [1] => 1 [2] => 1 [3] => 1 [4] => 1 )
// Разница множеств
$arr3 = array_diff_key($arr1, $arr2);
print_r($arr3);              // Array ( [3] => 1 )
// Пересечение множеств
$arr3 = array_intersect_key($arr1, $arr2);
print_r($arr3);              // Array ( [1] => 1 [2] => 1 )
// Проверка наличия элемента
var_dump(isset($arr1[1]));  // bool(true)
var_dump(isset($arr1[5]));  // bool(false)
```

5.6.19. Фильтрация массива

Функция `array_filter()` позволяет выполнить фильтрацию массива. Формат функции:

```
array_filter(array $array[, callable $callback=null[,
             int $flag=null]]) : array
```

В первом параметре указывается исходный массив, а во втором — функция обратного вызова. Если параметр `$flag` не указан, то функция `$callback` принимает в качестве параметра значение элемента. Если в параметре `$flag` указана константа `ARRAY_FILTER_USE_KEY`, то функция `$callback` принимает ключ элемента, а если задана константа `ARRAY_FILTER_USE_BOTH` — и значение, и ключ. Внутри функции нужно выполнить сравнение и вернуть логическое значение. В итоговый массив попадут только те элементы, для которых функция вернула значение `true`. Если функция обратного вызова не указана, то будут удалены все элементы, равные в логическом контексте значению `false`.

Функция `array_filter()` возвращает отфильтрованный массив. Исходный массив не изменяется. Ключи сохраняются:

```
$arr = array(1, 2, 3, 4, 5, 6);
$arr2 = array_filter($arr, function ($value, $key) {
   return ($value < 4 && $key != 1);
}, ARRAY_FILTER_USE_BOTH);
print_r( $arr2 ); // Array ( [0] => 1 [2] => 3 )
```

С помощью функции `array_map()` можно применить пользовательскую функцию ко всем элементам массива. Функция возвращает отфильтрованный массив. Формат функции:

```
array_map(callable $callback, array $array1[, array $...]) : array
```

Пример сложения значений двух списков:

```
$arr1 = array(1, 2, 3, 4, 5);
$arr2 = array(1, 2, 3, 4, 5);
$arr3 = array_map(function ($a, $b) {
   return ($a + $b);
}, $arr1, $arr2);
print_r( $arr3 );
// Array ( [0] => 2 [1] => 4 [2] => 6 [3] => 8 [4] => 10 )
```

5.7. Строки

Разрабатывая Web-сайты, часто приходится проводить манипуляции со *строками*. Поэтому необходимо знать и уметь использовать встроенные функции PHP, предназначенные для обработки строк. Например, перед добавлением сообщения в гостевую книгу можно удалить лишние пробелы и все теги из строки, добавить защитные слеши перед специальными символами или заменить их HTML-эквивалентами и т. д.

5.7.1. Инициализация строк

В языке PHP для хранения строк предназначен тип данных `string`:

```
$s = "string";
var_dump($s); // string(6) "string"
```

Строка может быть указана как внутри двойных кавычек, так и внутри апострофов:

```
$s = 'string';
var_dump($s); // string(6) "string"
```

Если в переменную нужно записать большой объем текста, это можно сделать способом, продемонстрированным в листинге 5.19.

Листинг 5.19. Запись в переменную большого объема текста

```
$s = <<<LABEL
Текст на
нескольких
строках
LABEL;
echo $s;
```

Результат выполнения:

```
Текст на
нескольких
строках
```

В этом примере многострочный текст располагается между метками `LABEL`. Название метки может быть любым допустимым идентификатором. Вторая (закрывающая) метка должна быть написана с начала строки, и после нее должна стоять точка с запятой. Других символов на этой строке быть не должно.

Название открывающей метки можно заключить в двойные кавычки или апострофы. В первом случае результат будет аналогичен строкам в двойных кавычках, а во втором — строкам внутри апострофов. Если кавычки не указаны, то результат будет аналогичен строкам в двойных кавычках:

```
$s1 = <<<"LABEL1"
Heredoc - аналог строки
в двойных кавычках\n
LABEL1;
echo $s1;

$s2 = <<<'LABEL2'
Nowdoc - аналог строки
в апострофах\n
LABEL2;
echo $s2;
```

Результат выполнения:

```
Heredoc - аналог строки
в двойных кавычках
Nowdoc - аналог строки
в апострофах\n
```

Обратите внимание: в последней строке результата специальный символ `\n` (перевод строки) был выведен как обычный символ, т. к. в строке, заключенной в апострофы, подстановка не осуществляется.

5.7.2. Специальные символы в строке

Внутри строки можно указать *специальные символы* — комбинации знаков, обозначающие служебные или непечатаемые символы, которые невозможно вставить обычным способом.

Приведем специальные символы, доступные в PHP в строках, ограниченных двойными кавычками:

- `\n` — перевод строки;
- `\r` — возврат каретки;
- `\t` — горизонтальная табуляция;

- `\v` — вертикальная табуляция;
- `\f` — перевод формата;
- `\"` — кавычка;
- `\$` — знак доллара;
- `\\` — обратная косая черта.

Кроме того, можно задавать символы восьмеричными или шестнадцатеричными кодами, записывая код символа после обратной косой черты или символов `\x`, соответственно:

```
// Восьмеричное значение
echo "\167"; // w
// Шестнадцатеричное значение
echo "\x77"; // w
```

С помощью последовательности `\u{N}` можно вставить в строку символ в кодировке Unicode:

```
echo "\u{005B}"; // [
echo "\u{005D}"; // ]
```

В строках, ограниченных апострофами, эти специальные символы не работают. В них доступны только два специальных символа:

- `\'` — апостроф;
- `\\` — обратная косая черта.

5.7.3. Подстановка значений переменных в строку

Очень часто необходимо сформировать строку, состоящую из имени переменной и (или) ее значения. Если написать:

```
$x = 10;
$s = "Значение равно $x";
```

то переменная `$s` будет содержать значение `"Значение равно 10"`, а если написать:

```
$x = 10;
$s = 'Значение равно $x';
```

то переменная `$s` будет содержать значение `"Значение равно $x"`. Помните, что строка в кавычках и строка в апострофах вернут разные результаты. В последнем случае, для того чтобы получить значение переменной, можно воспользоваться операцией конкатенации строк:

```
$x = 10;
$s = 'Значение равно ' . $x;
```

Рассмотрим еще один пример. Предположим, нужно объединить два слова в одно. Одно из слов задано с помощью переменной. Если написать:

```
$s = "авто";
$s2 = "$sтранспорт";
echo $s2; // $s2 = "" или Notice: Undefined variable
```

то переменная `$s2` будет содержать пустую строку, т. к. переменная `$sтранспорт` не определена. В этом случае можно воспользоваться следующими способами:

- использовать конкатенацию строк:

```
$s = "авто";
$s2 = $s . "транспорт";
echo $s2; // автотранспорт
```

- указать имя переменной в фигурных скобках так:

```
$s = "авто";
$s2 = "${s}транспорт";
echo $s2; // автотранспорт
```

  или так:

```
$s = "авто";
$s2 = "{$s}транспорт";
echo $s2; // автотранспорт
```

Если нужно вставить в строку имя переменной, то следует перед символом `$` указать обратную косую черту:

```
$x = 10;
$s = "Значение переменной \$x равно $x";
echo $s; // Значение переменной $x равно 10
```

5.7.4. Конкатенация строк

Оператор `.` (точка) осуществляет конкатенацию строк, т. е. соединяет их в одну строку:

```
$s = "Строка1" . "Строка2";
echo $s;            // Строка1Строка2
```

Можно также воспользоваться оператором `.=`:

```
$s = "Строка1";
$s .= "Строка2";
echo $s;            // Строка1Строка2
```

С помощью конкатенации строк можно преобразовать любой тип данных в строку:

```
$x = 10;
var_dump($x . ""); // string(2) "10"
```

Аналогичный результат будет достигнут при указании переменной внутри двойных кавычек:

```
$x = 10;
var_dump("$x");    // string(2) "10"
```

5.7.5. Строка в обратных кавычках. Запуск внешних программ

Если содержимое строки заключить в обратные кавычки, то это позволит запустить внешнюю программу и присвоить переменной результат ее работы (листинг 5.20).

Листинг 5.20. Запуск внешней программы

```
<?php
$v = `dir`;
echo '<textarea cols="70" rows="30">';
echo iconv("cp866", "UTF-8", $v);
echo '</textarea>';
```

Этот код выведет содержимое каталога C:\xampp\htdocs. При выводе используется кодировка DOS (кодовая страница 866), поэтому русские буквы будут искажены. Чтобы избежать этого, мы преобразуем кодировку с помощью функции `iconv()`.

Аналогичный результат можно получить с помощью функции `exec()` (листинг 5.21). Формат функции:

```
exec(string $command[, array &$output=null[,
     int &$return_var=null]]) : string
```

Листинг 5.21. Запуск внешней программы с помощью функции `exec()`

```
<?php
$arr = [];
exec("dir", $arr);
$v = implode("\n", $arr);
echo '<textarea cols="70" rows="30">';
echo iconv("cp866", "UTF-8", $v);
echo '</textarea>';
```

5.7.6. Обращение к отдельному символу в строке

Строка в языке PHP является массивом байтов, поэтому к любому символу строки можно обратиться как к элементу массива, — достаточно указать его индекс в квадратных или фигурных скобках. Нумерация начинается с нуля. Можно как получить символ по индексу, так и изменить его значение:

```
$s = "string";
echo $s[0];       // Выведет: s
echo $s{1};       // Выведет: t
$s[3] = "o";
echo $s;          // Выведет: strong
```

Допускается указание индекса с отрицательным значением. В этом случае указанное значение вычитается из длины строки. Вот пример получения последнего символа строки:

```
$s = "string";
echo $s[-1];             // Выведет: g
echo $s[strlen($s) - 1]; // Выведет: g
```

Для работы с отдельными символами предназначены следующие функции:

- `chr(<ASCII-код символа>)` — возвращает символ по указанному коду:

  ```
  echo chr(81);  // Выведет: Q
  ```

- `ord(<Символ>)` — возвращает ASCII-код указанного символа:

  ```
  echo ord("Q"); // Выведет: 81
  ```

При работе с многобайтовыми кодировками доступ по индексу вернет лишь часть символа (один байт), а не символ целиком. Все указанные ранее примеры работают с кодировкой UTF-8 лишь по причине того, что английские буквы кодируются одним байтом. Если же мы попробуем обратиться по индексу при использовании строки с русскими буквами, которые кодируются двумя байтами, то получим какой-то непонятный символ (половинку от русской буквы). В этом случае для получения отдельного символа нужно воспользоваться функцией `mb_substr()`:

```
$s = "строка"; // Кодировка UTF-8
echo ord($s[0]);                    // Выведет: 209
echo ord($s[1]);                    // Выведет: 129
echo mb_substr($s, 0, 1, 'UTF-8'); // Выведет: с
echo mb_substr($s, 1, 1, 'UTF-8'); // Выведет: т
```

5.7.7. Строки в кодировке UTF-8

Итак, в предыдущем разделе мы столкнулись с первой особенностью при использовании кодировки UTF-8 — строка в этой кодировке является массивом байтов, а не массивом символов. При доступе по индексу мы получим один байт из символа. Тем не менее, к символам кодировки ASCII мы можем обратиться как к элементам массива, т. к. они кодируются одним байтом:

```
$s = "string";
echo $s[0];      // Получили букву s
$s = "строка";
echo $s[0];      // Получили лишь один байт, а не букву с!
```

В однобайтовых кодировках символ кодируется одним байтом. Первые 7 битов позволяют закодировать 128 символов, соответствующих кодировке ASCII. Символы, имеющие код меньше `33`, являются специальными, — например, нулевой символ, символ перевода строки, табуляция и т. д. Получить остальные символы позволяет следующий код:

```
for ($i = 33; $i < 128; $i++) {
   echo $i . " =&gt; " . chr($i) . "<br>\n";
}
```

Коды этих символов одинаковы практически во всех однобайтовых кодировках. Восьмой же бит предназначен для кодирования символов национальных алфавитов. То есть однобайтовые кодировки позволяют закодировать всего 256 символов.

К любому символу строки в однобайтовой кодировке (например, windows-1251 или KOI8-R) можно обратиться как к элементу массива. Достаточно указать его индекс в квадратных скобках. Нумерация начинается с нуля:

```
$s = 'строка'; // Кодировка windows-1251
echo $s[0];    // Выведет: с
```

В кодировке UTF-8 один символ может кодироваться несколькими байтами. Первые 128 символов соответствуют кодировке ASCII и кодируются всего одним байтом. Остальные символы кодируются переменным числом байтов от двух до шести (на практике только до четырех). Буквы русского алфавита и некоторых других европейских языков кодируются двумя байтами. По этой причине использовать обычные строковые функции нельзя, т. к. они работают в большинстве случаев с однобайтовыми кодировками.

Для работы с многобайтовыми кодировками (в том числе и с UTF-8) необходимо использовать функции из библиотек `iconv` и `php_mbstring`. Подключение библиотеки `php_mbstring` производится в конфигурационном файле php.ini с помощью следующей инструкции (перед инструкцией не должно быть точки с запятой, являющейся комментарием):

```
extension=php_mbstring.dll
```

В PHP 7.2 инструкция имеет следующий формат:

```
extension=mbstring
```

В пакете XAMPP библиотека `php_mbstring` подключается по умолчанию.

Все функции из этих библиотек содержат параметр `$encoding`, в котором нужно указать кодировку строки. Например, функция `mb_substr()`, с помощью которой мы получали отдельный символ, имеет следующий формат:

```
mb_substr(string $str, int $start[, int $length=null[,
          string $encoding=mb_internal_encoding()]]) : string
```

Обратите внимание: параметр `$encoding` является необязательным. Если параметр не указан, то:

- при вызове функций, начинающихся с префикса `mb_`, используется значение, указанное в функции `mb_internal_encoding()`. До версии 5.6 также использовалось значение директивы `mbstring.internal_encoding`. В PHP 7.1 установка значения этой директивы выводит предупреждающее сообщение, что директива устарела, и следует использовать `internal_encoding` или `default_charset`. Однако функция `mb_internal_encoding()` такого сообщения не выводит, но при этом

берет значение из директивы `mbstring.internal_encoding`. Проблема в том, что все функции из этой библиотеки получают значение кодировки по умолчанию из функции `mb_internal_encoding()`, а не из директивы `default_charset`. Поэтому установка значений в директивах `internal_encoding` и `default_charset` не меняет кодировку по умолчанию, хотя вроде как должна;

- при вызове функций, начинающихся с префикса `iconv_`, используется значение директивы `internal_encoding` или `default_charset` (до версии 5.6 использовалось значение директивы `iconv.internal_encoding` или значение, указанное в функции `iconv_set_encoding()`).

Иными словами, в программе мы можем:

- либо указать кодировку в функции:

```
$s = 'строка';
echo mb_substr($s, 0, 1, 'UTF-8'); // Выведет: с
```

- либо настроить кодировку в начале программы и в функции ее не указывать:

```
mb_internal_encoding('UTF-8');
$s = 'строка';
echo mb_substr($s, 0, 1); // Выведет: с
```

- либо настроить сервер и PHP на нужную кодировку и вообще кодировку не указывать.

Проверить, какая кодировка настроена по умолчанию, позволяет следующий код:

```
echo ini_get('default_charset'); // UTF-8
echo mb_internal_encoding();     // UTF-8
```

Напомним, что в *главе 4* мы настроили сервер и PHP на кодировку UTF-8, поэтому в обоих случаях вы должны получить именно эту кодировку в качестве результата. Кроме того, в *разд. 4.3* мы создали каталог C:\xampp\htdocs\cp1251 и настроили для него кодировку по умолчанию windows-1251. Поэтому результат выполнения программы из этого каталога должен быть следующим:

```
echo ini_get('default_charset'); // windows-1251
```

Каким будет возвращаемое значение функции `mb_internal_encoding()` при работе с кодировкой windows-1251 не важно, т. к. функции из библиотеки `php_mbstring` в этом случае практически не используются.

Кодировкой UTF-8 в этом издании книги мы пользуемся по умолчанию. Однако в этом разделе мы будем рассматривать и функции, предназначенные для работы с однобайтовыми кодировками. Если указан комментарий:

```
// Кодировка файла windows-1251
```

то такой код нужно запускать именно из каталога C:\xampp\htdocs\cp1251. Причем не забудьте и сам файл с программой сохранить в кодировке windows-1251.

Если комментарий не указан, то используется кодировка UTF-8. Еще раз напомним, что в этом случае при сохранении файла с программой в Notepad++ в меню **Коди-**

ровки должен быть установлен флажок **Кодировать в UTF-8 (без BOM)**. В противном случае метка порядка байтов (сокращенно, BOM) может стать причиной ошибок при формировании заголовков ответа сервера из программы.

5.7.8. Преобразование кодировок

Для преобразования кодировок используется функция `mb_convert_encoding()` со следующим форматом вызова:

```
mb_convert_encoding(<Исходная строка>, <Нужная кодировка>[,
                    <Исходная кодировка>])
```

Вот пример преобразования строки из кодировки UTF-8 в windows-1251:

```
$str = mb_convert_encoding("Строка", "windows-1251", "UTF-8");
echo strlen($str); // 6
```

Получить список поддерживаемых кодировок можно так:

```
$arr = mb_list_encodings();
sort($arr, SORT_NATURAL | SORT_FLAG_CASE);
print_r($arr);
```

Функция `iconv()` также преобразовывает символы строки из одной кодировки в другую. Формат функции:

```
iconv(<Исходная кодировка>, <Нужная кодировка>[<Флаг>],
      <Исходная строка>)
```

Пример преобразования строки из кодировки windows-1251 в UTF-8:

```
// Кодировка файла windows-1251
$str = iconv("windows-1251", "UTF-8", "Строка");
echo strlen($str); // 12
```

Необязательный параметр `<Флаг>` может принимать следующие значения:

- `//TRANSLIT` — если символа нет в нужной кодировке, он заменяется одним или несколькими аналогами;
- `//IGNORE` — символы, которых нет в нужной кодировке, будут опущены.

Зачем нужен этот параметр? Если мы преобразовываем кодировку windows-1251 в UTF-8, то в этом параметре нет необходимости. А вот если наоборот, то может возникнуть ситуация, что символа нет в нужной кодировке, т. к. кодировка UTF-8 позволяет хранить несколько тысяч символов, а кодировка windows-1251 только 256 символов. Если не указать этот параметр, то строка будет обрезана до первого недопустимого символа:

```
$str = iconv("UTF-8", "windows-1251//IGNORE", "Строка");
echo strlen($str); // 6
```

> ***Внимание!***
>
> Порядок следования параметров в функции `mb_convert_encoding()` отличается от порядка в функции `iconv()`.

С помощью функции `convert_cyr_string()` можно преобразовать строку из одной однобайтовой кодировки в другую. Формат функции:

```
convert_cyr_string(<Исходная строка>, <Исходная кодировка>,
                   <Нужная кодировка>)
```

Значения параметров `<Исходная кодировка>` и `<Нужная кодировка>`:

- `a` или `d` — кодировка x-cp866;
- `i` — кодировка iso8859-5;
- `k` — кодировка KOI8-R;
- `m` — кодировка x-mac-cyrillic;
- `w` — кодировка windows-1251.

Пример вызова функции:

```
// Кодировка файла windows-1251
$str = "уФТПЛБ";
echo convert_cyr_string($str, "k", "w"); // Выведет: Строка
```

5.7.9. Определение длины строки

Получить длину строки позволяют следующие функции:

- `mb_strlen(<Строка>[, <Кодировка>])` — возвращает число символов в строке:

  ```
  $str = 'Строка';
  echo mb_strlen($str, 'UTF-8');    // 6
  ```

- `iconv_strlen(<Строка>[, <Кодировка>])` — возвращает число символов в строке:

  ```
  $str = 'Строка';
  echo iconv_strlen($str, 'UTF-8'); // 6
  ```

- `strlen(<Строка>)` — возвращает число байтов в строке. Так как в однобайтовых кодировках один символ описывается одним байтом, функция `strlen()` возвращает число символов. Для многобайтовых кодировок функция возвращает именно число байтов:

  ```
  $str = 'Строка UTF-8';
  echo strlen($str);                // 18
  $str = iconv('UTF-8', 'windows-1251', $str);
  echo strlen($str);                // 12
  ```

  Почему же мы получили 18 байтов, а не 24? Все дело в том, что в кодировке UTF-8 первые 128 символов кодируются одним байтом, а все последующие символы — несколькими байтами. Каждый символ в слове `Строка` занимает по два байта, а в последующей части строки (`UTF-8`) каждый символ занимает один байт. Итого: 6 умножить на 2 плюс 6 равно 18 байтов.

> ***Внимание!***
>
> Если в файле *php.ini* директива `mbstring.func_overload` равна `2` или `7`, то функция `strlen()` полностью эквивалентна функции `mb_strlen()`. Это означает, что функция

`strlen()` будет возвращать число символов, а не байтов. Начиная с версии 7.2, механизм перегрузки функций объявлен устаревшим и не рекомендуется к использованию.

5.7.10. Настройка локали

Локалью называют совокупность локальных настроек системы. Настройки локали для разных стран различаются. Например, в одной стране принято десятичный разделитель вещественных чисел выводить в виде точки, а в другой — в виде запятой. Настройки локали влияют также на работу с датой и со строками — например, изменение регистра символов при неправильно настроенной локали может выполняться некорректно.

Для установки локали используется функция `setlocale()`. Формат функции:

```
setlocale(<Категория>, <Локаль>)
```

Параметр `<Категория>` может принимать следующие значения:

- `LC_ALL` — устанавливает локаль для всех режимов;
- `LC_COLLATE` — для сравнения строк;
- `LC_CTYPE` — для перевода символов в нижний или верхний регистр;
- `LC_MONETARY` — для отображения денежных единиц;
- `LC_NUMERIC` — для форматирования вещественных чисел;
- `LC_TIME` — для форматирования вывода даты и времени.

Настройка локали различна в UNIX и Windows. В UNIX при настройке русской локали используется следующий синтаксис:

```
setlocale(LC_ALL, 'ru_RU');
```

А в Windows такой:

```
setlocale(LC_ALL, 'Russian_Russia');
```

Чтобы не было зависимости от операционной системы, достаточно привести сразу несколько локалей через запятую:

```
setlocale(LC_ALL, 'ru_RU', 'Russian_Russia');
```

или в виде массива:

```
setlocale(LC_ALL, array('ru_RU', 'Russian_Russia'));
```

Если установить локаль удалось, то функция вернет имя установленной локали, в противном случае — значение `false`.

После имени локали через точку можно задать кодировку:

```
setlocale(LC_ALL, 'ru_RU.CP1251', 'Russian_Russia.1251');
```

Если во втором параметре указана пустая строка или значение `null`, то настройки локали будут взяты из переменных окружения. Если во втором параметре задано значение `"0"`, то функция вернет текущее значение локали, не изменяя его:

```
echo setlocale(LC_ALL, "0");
// LC_COLLATE=C;LC_CTYPE=Russian_Russia.1251;
// LC_MONETARY=C;LC_NUMERIC=C;LC_TIME=C
printf("%.2f\n", 10.5125484);        // 10.51
setlocale(LC_ALL, 'ru_RU', 'Russian_Russia');
echo setlocale(LC_ALL, "0");
// Russian_Russia.1251
printf("%.2f", 10.5125484);          // 10,51
```

5.7.11. Изменение регистра символов

Для изменения регистра символов в строке предназначены следующие функции:

- `mb_strtoupper(<Строка>[, <Кодировка>])` — приводит все символы строки к верхнему регистру:

  ```
  $str = 'строка СТРОКА СтРоКа';
  echo mb_strtoupper($str, 'UTF-8'); // СТРОКА СТРОКА СТРОКА
  ```

- `strtoupper()` — преобразует все буквы в строке к верхнему регистру:

  ```
  // Кодировка файла windows-1251
  setlocale(LC_CTYPE, 'ru_RU', 'Russian_Russia');
  $str = 'строка СТРОКА СтРоКа';
  echo strtoupper($str); // СТРОКА СТРОКА СТРОКА
  ```

  По умолчанию работает только с однобайтовыми кодировками;

- `mb_strtolower(<Строка>[, <Кодировка>])` — приводит все символы строки к нижнему регистру:

  ```
  $str = 'строка СТРОКА СтРоКа';
  echo mb_strtolower($str, 'UTF-8'); // строка строка строка
  ```

- `strtolower()` — преобразует все буквы в строке к нижнему регистру:

  ```
  // Кодировка файла windows-1251
  setlocale(LC_CTYPE, 'ru_RU', 'Russian_Russia');
  $str = 'строка СТРОКА СтРоКа';
  echo strtolower($str); // строка строка строка
  ```

  По умолчанию работает только с однобайтовыми кодировками;

- `mb_convert_case(<Строка>, <Режим>[, <Кодировка>])` — преобразует регистр символов в зависимости от значения параметра `<Режим>`, который может принимать следующие значения:

  - `MB_CASE_UPPER` — приводит все символы строки к верхнему регистру;
  - `MB_CASE_LOWER` — приводит все символы строки к нижнему регистру;
  - `MB_CASE_TITLE` — приводит первые символы всех слов к верхнему регистру.

  Примеры:

  ```
  $str = 'строка СТРОКА СтРоКа';
  echo mb_convert_case($str, MB_CASE_UPPER, 'UTF-8');
  ```

```
// СТРОКА СТРОКА СТРОКА
echo '<br>';
echo mb_convert_case($str, MB_CASE_LOWER, 'UTF-8');
// строка строка строка
echo '<br>';
echo mb_convert_case($str, MB_CASE_TITLE, 'UTF-8');
// Строка Строка Строка
```

- `ucfirst()` — преобразует первую букву строки к верхнему регистру:

```
// Кодировка файла windows-1251
setlocale(LC_CTYPE, 'ru_RU', 'Russian_Russia');
$str = 'строка СТРОКА СтРоКа';
echo ucfirst($str);              // Строка СТРОКА СтРоКа
echo ucfirst(strtolower($str)); // Строка строка строка
```

  Работает только с однобайтовыми кодировками;

- `ucwords()` — преобразует первые буквы всех слов к верхнему регистру. Работает только с однобайтовыми кодировками. Формат функции:

```
ucwords(string $str[, string $delimiters]): string
```

  В параметре `$delimiters` можно указать символы, которые допускаются перед словом. Значение по умолчанию: `" \t\r\n\f\v"`:

```
// Кодировка файла windows-1251
setlocale(LC_CTYPE, 'ru_RU', 'Russian_Russia');
$str = 'строка СТРОКА СтРоКа';
echo ucwords($str);              // Строка СТРОКА СтРоКа
echo ucwords(strtolower($str)); // Строка Строка Строка
$str = 'строка+строка+строка';
echo ucwords($str, "+");        // Строка+Строка+Строка
```

> ***Внимание!***
>
> Если в файле *php.ini* директива `mbstring.func_overload` равна `2` или `7`, то функция `strtoupper()` будет эквивалентна функции `mb_strtoupper()`, а функция `strtolower()` — функции `mb_strtolower()`. Начиная с версии 7.2, механизм перегрузки функций объявлен устаревшим и не рекомендуется к использованию.

5.7.12. Получение фрагмента строки

Получить фрагмент строки позволяют следующие функции:

- `mb_substr()` — возвращает подстроку указанной длины, начиная с заданной позиции. Если длина не указана или задано значение `null`, то возвращается подстрока, начиная с заданной позиции и до конца строки. Формат функции:

```
mb_substr(<Строка>, <Начальная позиция>[, <Длина>[, <Кодировка>]])
```

  Пример:

```
$str = 'Строка';
echo mb_substr($str, 0, 1, 'UTF-8');     // С
```

```
echo mb_substr($str, 2, 3, 'UTF-8');    // рок
echo mb_substr($str, -3, 3, 'UTF-8');   // ока
echo mb_substr($str, 0, null, 'UTF-8'); // Строка
```

- `iconv_substr()` — возвращает подстроку указанной длины, начиная с заданной позиции. Если длина не указана, то возвращается подстрока, начиная с заданной позиции и до конца строки. Формат функции:

  ```
  iconv_substr(<Строка>, <Начальная позиция>[, <Длина>[, <Кодировка>]])
  ```

  Пример:

  ```
  $str = 'Строка';
  echo iconv_substr($str, 0, 1, 'UTF-8');  // С
  echo iconv_substr($str, 2, 3, 'UTF-8');  // рок
  echo iconv_substr($str, -3, 3, 'UTF-8'); // ока
  ```

- `substr()` — возвращает подстроку указанной длины, начиная с заданной позиции. Если длина не указана, то возвращается подстрока, начиная с заданной позиции и до конца строки. По умолчанию работает только с однобайтовыми кодировками. Формат функции:

  ```
  substr(<Строка>, <Начальная позиция>[, <Длина>])
  ```

  Пример:

  ```
  // Кодировка файла windows-1251
  $str = 'Строка';
  echo substr($str, 0, 1);  // С
  echo substr($str, 1);     // трока
  echo substr($str, -1, 1); // а
  ```

> **Внимание!**
>
> Если в файле *php.ini* директива `mbstring.func_overload` равна `2` или `7`, то функция `substr()` будет полностью эквивалентна функции `mb_substr()`. Начиная с версии 7.2, механизм перегрузки функций объявлен устаревшим и не рекомендуется к использованию.

5.7.13. Сравнение строк

Если попытаться сравнить две строки, содержащие числа, или строку и число с помощью операторов `==` и `!=`, то строки будут преобразованы в числа. После этого будут сравниваться числа, а не строки:

```
var_dump("10 a" == 10);       // bool(true)
var_dump("10" == "010");      // bool(true)
```

Если попытаться использовать оператор `<=>`, то результат будет таким же:

```
var_dump("10 a" <=> 10);       // int(0)
var_dump("10" <=> "010");      // int(0)
```

Чтобы преобразования типов не происходило, нужно использовать операторы сравнения `===` и `!==`:

```
var_dump("10 a" === 10);       // bool(false)
var_dump("10" === "010");      // bool(false)
var_dump("10 a" === "10 a");   // bool(true)
```

Для сравнения строк предназначены следующие функции:

- `strcmp(<Строка1>, <Строка2>)` — сравнивает две строки. Возвращает одно из трех значений:
  - `0` — если строки равны;
  - положительное число — если `<Строка1>` больше `<Строки2>`;
  - отрицательное число — если `<Строка1>` меньше `<Строки2>`.

  Пример:

  ```
  $str1 = "Строка1";
  $str2 = "Строка2";
  echo strcmp($str1, $str2); // -1
  ```

  Функция зависит от регистра символов. Если нужно сделать сравнение не зависящим от регистра, то следует предварительно изменить регистр символов:

  ```
  $str1 = "строка";
  $str2 = "СТРОКА";
  var_dump( strcmp($str1, $str2) ); // int(1)
  // Преобразуем к нижнему регистру
  $s1 = mb_strtolower($str1, 'UTF-8');
  $s2 = mb_strtolower($str2, 'UTF-8');
  // Сравниваем
  var_dump( strcmp($s1, $s2) );     // int(0)
  ```

- `strcoll(<Строка1>, <Строка2>)` — сравнивает строки на основе локализации. Зависит от регистра символов. Если локаль не настроена, то эта функция эквивалентна функции `strcmp()`:

  ```
  // Кодировка файла windows-1251
  setlocale(LC_ALL, 'ru_RU', 'Russian_Russia');
  $str1 = "Строка1";
  $str2 = "Строка2";
  echo strcoll($str1, $str2); // -1
  ```

  Работает только с однобайтовыми кодировками;

- `strncmp(<Строка1>, <Строка2>, <Длина>)` — сравнивает указанное число первых байтов строк. Зависит от регистра символов:

  ```
  // Кодировка файла windows-1251
  $str1 = "Строка1";
  $str2 = "Строка2";
  echo strncmp($str1, $str2, 6); // 0
  echo strncmp($str1, $str2, 7); // -1
  ```

- `substr_compare()` — сравнивает фрагмент строки `$main_str` длиной `$length`, начиная со смещения `$offset`, со строкой `$str`. Если в параметре `$case_`

`insensitivity` указано значение `true`, то сравнение будет выполняться без учета регистра символов. Работает только с однобайтовыми кодировками. Формат функции:

```
substr_compare(string $main_str, string $str, int $offset[,
       int $length[, bool $case_insensitivity=false]]) : int
```

Пример:

```
// Кодировка файла windows-1251
setlocale(LC_ALL, 'ru_RU', 'Russian_Russia');
$str1 = "Строка1";
$str2 = "строка2";
echo substr_compare($str1, $str2, 0, 6);       // -1
echo substr_compare($str1, $str2, 0, 6, true); // 0
echo substr_compare($str1, $str2, 0, 7, true); // -1
```

5.7.14. Поиск в строке

Для поиска в строке предусмотрены следующие функции:

- `mb_strpos()` — ищет подстроку в строке. Возвращает номер позиции, с которой начинается вхождение подстроки в строку. Если подстрока в строку не входит, то функция возвращает `false`. Функция зависит от регистра символов и имеет следующий формат:

```
mb_strpos(<Строка>, <Подстрока>[,
          <Начальная позиция поиска>[, <Кодировка>]])
```

  Если начальная позиция не указана, то поиск будет осуществляться с начала строки:

```
echo mb_strpos('Привет', 'ри', 0, 'UTF-8'); // Выведет: 1
mb_internal_encoding('UTF-8'); // Установка кодировки
if (mb_strpos('Привет', 'При') !== false)
   echo 'Найдено'; // Выведет: Найдено
else echo 'Не найдено';
```

- `iconv_strpos()` — ищет подстроку в строке. Возвращает номер позиции, с которой начинается вхождение подстроки в строку. Если подстрока в строку не входит, то функция возвращает `false`. Функция зависит от регистра символов. Если начальная позиция не указана, то поиск будет производиться с начала строки. Формат функции:

```
iconv_strpos(<Строка>, <Подстрока>[,
             <Начальная позиция поиска>[, <Кодировка>]])
```

  Примеры:

```
echo iconv_strpos('Привет', 'ри', 0, 'UTF-8'); // Выведет: 1
if (iconv_strpos('Привет', 'При', 0, 'UTF-8') !== false)
   echo 'Найдено'; // Выведет: Найдено
else echo 'Не найдено';
```

- `strpos()` — ищет подстроку в строке. Возвращает номер позиции, с которой начинается вхождение подстроки в строку. Если подстрока в строку не входит, то функция возвращает `false`. Функция зависит от регистра символов. По умолчанию работает только с однобайтовыми кодировками. Формат функции:

  ```
  strpos(<Строка>, <Подстрока>[, <Начальная позиция поиска>])
  ```

  Если начальная позиция не указана, то поиск будет проводиться с начала строки:

  ```
  // Кодировка файла windows-1251
  if (strpos('Привет', 'При') !== false)
     echo 'Найдено'; // Выведет: Найдено
  else echo 'Не найдено';
  ```

  > ***Внимание!***
  >
  > Если в файле *php.ini* директива `mbstring.func_overload` равна `2` или `7`, то функция `strpos()` будет полностью эквивалентна функции `mb_strpos()`. Начиная с версии 7.2, механизм перегрузки функций объявлен устаревшим и не рекомендуется к использованию.

- `mb_stripos()` — ищет подстроку в строке. Возвращает номер позиции, с которой начинается вхождение подстроки в строку. Если подстрока в строку не входит, то функция возвращает `false`. В отличие от функции `mb_strpos()`, не зависит от регистра символов. Формат функции:

  ```
  mb_stripos(<Строка>, <Подстрока>[,
             <Начальная позиция поиска>[, <Кодировка>]])
  ```

  Если начальная позиция не указана, то поиск будет производиться с начала строки:

  ```
  echo mb_stripos('Привет', 'РИ', 0, 'UTF-8'); // Выведет: 1
  mb_internal_encoding('UTF-8');      // Установка кодировки
  echo mb_stripos('Привет', 'РИ');    // Выведет: 1
  ```

- `stripos()` — ищет подстроку в строке. Возвращает номер позиции, с которой начинается вхождение подстроки в строку. Если подстрока в строку не входит, то функция возвращает `false`. В отличие от функции `strpos()`, не зависит от регистра символов. По умолчанию работает только с однобайтовыми кодировками. Формат функции:

  ```
  stripos(<Строка>, <Подстрока>[, <Начальная позиция поиска>])
  ```

  Пример:

  ```
  // Кодировка файла windows-1251
  setlocale(LC_ALL, 'ru_RU', 'Russian_Russia');
  echo stripos('Привет', 'РИ', 0); // Выведет: 1
  ```

  > ***Внимание!***
  >
  > Если в файле *php.ini* директива `mbstring.func_overload` равна `2` или `7`, то функция `stripos()` будет полностью эквивалентна функции `mb_stripos()`. Начиная с версии 7.2, механизм перегрузки функций объявлен устаревшим и не рекомендуется к использованию.

- `mb_strrpos()` — ищет подстроку в строке. Возвращает позицию последнего вхождения подстроки в строку. Если подстрока в строку не входит, то функция возвращает `false`. Функция зависит от регистра символов. Формат функции:

```
mb_strrpos(<Строка>, <Подстрока>[,
           <Начальная позиция поиска>[, <Кодировка>]])
```

Пример:

```
echo mb_strrpos('ерпарверпр', 'ер', 0, 'UTF-8'); // 6
```

- `iconv_strrpos()` — ищет подстроку в строке. Возвращает позицию последнего вхождения подстроки в строку. Если подстрока в строку не входит, то функция возвращает `false`. Функция зависит от регистра символов. Формат функции:

```
iconv_strrpos(<Строка>, <Подстрока>[, <Кодировка>])
```

Пример:

```
echo iconv_strrpos('ерпарверпр', 'ер', 'UTF-8'); // 6
```

- `mb_strripos()` — ищет подстроку в строке. Возвращает позицию последнего вхождения подстроки в строку. Если подстрока в строку не входит, то функция возвращает `false`. В отличие от функции `mb_strrpos()`, не зависит от регистра символов. Формат функции:

```
mb_strripos(<Строка>, <Подстрока>[,
           <Начальная позиция поиска>[, <Кодировка>]])
```

Пример:

```
echo mb_strripos('ерпарверпр', 'ЕР', 0, 'UTF-8'); // 6
```

- `mb_substr_count()` — возвращает число вхождений подстроки в строку. Функция зависит от регистра символов. Формат функции:

```
mb_substr_count(<Строка>, <Подстрока>[, <Кодировка>])
```

Пример:

```
echo mb_substr_count('ерпаерпр', 'ер', 'UTF-8'); // 2
```

5.7.15. Замена в строке

Для замены в строке предназначены следующие функции:

- `str_replace()` — заменяет все вхождения подстроки в строке на другую подстроку и возвращает результат в виде новой строки (или массива). Функция не изменяет исходную строку и зависит от регистра символов. Формат функции:

```
str_replace(<Подстрока для замены>, <Новая подстрока>,
            <Строка>[, <Число произведенных замен>])
```

Если в необязательном четвертом параметре указать переменную, то в ней будет сохранено число проведенных замен:

```
$str = 'Привет, Петя';
$count = 0;
```

```
$str = str_replace('Петя', 'Вася', $str, $count);
echo $str;   // Выведет: Привет, Вася
echo $count; // Выведет: 1
```

В качестве параметра можно также передать массив:

```
$arr = array('!', '@', '#', '$', '%', '^', '&', '*',
             '(', ')', '_', '+', '=', '.');
echo str_replace($arr, '', 'Текст !@#$%^&*()_+=. текст');
// Выведет: Текст  текст
```

- trim(<Строка>[, <Символы>]) — удаляет пробельные (или указанные во втором параметре) символы в начале и конце строки. Пробельными символами считаются: пробел, символ перевода строки (\n), символ возврата каретки (\r), символы горизонтальной (\t) и вертикальной (\v) табуляции и нулевой символ (\0):

```
$str = ' Строка ';
$str = trim($str);
echo "'$str'";           // Выведет: 'Строка'
echo trim($str, " aC");  // Выведет: трок
```

- ltrim(<Строка>[, <Символы>]) — удаляет пробельные (или указанные во втором параметре) символы в начале строки:

```
$str = ' Строка ';
$str = ltrim($str);
echo "'$str'";           // Выведет: 'Строка '
echo ltrim($str, " aC"); // Выведет: трока
```

- rtrim(<Строка>[, <Символы>]) — удаляет пробельные (или указанные во втором параметре) символы в конце строки:

```
$str = ' Строка ';
$str = rtrim($str);
echo "'$str'";           // Выведет: ' Строка'
echo rtrim($str, " aC"); // Выведет:  Строк
```

- chop(<Строка>[, <Символы>]) — удаляет пробельные (или указанные во втором параметре) символы в конце строки:

```
$str = ' Строка ';
$str = chop($str);
echo "'$str'";          // Выведет: ' Строка'
echo chop($str, " aC"); // Выведет:  Строк
```

- strip_tags(<Строка>[, <Теги>]) — удаляет из строки все HTML- и PHP-теги, за исключением указанных во втором параметре:

```
$str = '<span style="color: red"><b>Строка</b></span>';
$str1 = strip_tags($str);
$str2 = strip_tags($str, '<b>');
echo $str1; // Выведет: Строка
echo $str2; // Выведет: <b>Строка</b>
```

Следует заметить, что функция `strip_tags()` работает не совсем корректно. Если в строке встретится открывающая угловая скобка (<), а за ней сразу другой символ, то будет удален весь фрагмент от скобки до конца строки:

```
$str = '5<10 Эта строка будет удалена!';
echo strip_tags($str); // Выведет: 5
```

- `addslashes()` — добавляет обратную косую черту для защиты специальных символов:

```
$str = '"Волга", "Москвич", "Жигули"';
$str = addslashes($str);
echo $str; // Выведет: \"Волга\", \"Москвич\", \"Жигули\"
```

- `stripslashes()` — удаляет символы обратной косой черты:

```
$str = '\\"Волга\\", \\"Москвич\\", \\"Жигули\\"';
$str = stripslashes($str);
echo $str; // Выведет: "Волга", "Москвич", "Жигули"
```

- `htmlspecialchars()` — заменяет специальные символы их HTML-эквивалентами. Формат функции:

```
htmlspecialchars(string $string,
       int $flags=ENT_COMPAT | ENT_HTML401,
       string $encoding=ini_get('default_charset'),
       bool $double_encode=true) : string
```

Необязательный параметр `$flags` задает режим преобразования двойных и одинарных кавычек. Может принимать следующие значения (или их комбинацию):

  - `ENT_COMPAT` — преобразуются только двойные кавычки (значение по умолчанию);
  - `ENT_QUOTES` — преобразуются и двойные, и одинарные кавычки;
  - `ENT_NOQUOTES` — двойные и одинарные кавычки не заменяются;
  - `ENT_IGNORE` — отбрасывает некорректные кодовые последовательности (не рекомендуется использовать это значение в целях безопасности);
  - `ENT_SUBSTITUTE` — заменяет некорректные кодовые последовательности символом `\u{FFFD}` в случае применения UTF-8 или `&#xFFFD;` при использовании другой кодировки;
  - `ENT_HTML401` — обрабатывает код по правилам HTML 4.01 (значение по умолчанию);
  - `ENT_HTML5` — обрабатывает код по правилам HTML 5;
  - `ENT_XML1` — обрабатывает код по правилам XML 1;
  - `ENT_XHTML` — обрабатывает код по правилам XHTML.

В необязательном параметре `$encoding` можно указать кодировку. В PHP 7 по умолчанию используется значение директивы `default_charset`. В предыдущих

версиях значение по умолчанию часто менялось, поэтому кодировку лучше всегда указывать явным образом.

Если в необязательном параметре `$double_encode` указано значение `false`, то существующие в строке HTML-эквиваленты повторно преобразовываться не будут:

```
$str = '"Волга", "Москвич" &quot;';
$str2 = htmlspecialchars($str, ENT_COMPAT | ENT_HTML5,
                        'UTF-8');
echo $str2;
// &quot;Волга&quot;, &quot;Москвич&quot; &amp;quot;
$str = htmlspecialchars($str, ENT_COMPAT | ENT_HTML5, UTF-8', false);
echo $str;
// &quot;Волга&quot;, &quot;Москвич&quot; &quot;
```

С помощью функции `htmlspecialchars_decode()` можно выполнить обратное преобразование:

```
$str = '&quot;Волга&quot;, &quot;Москвич&quot;';
$str = htmlspecialchars_decode($str, ENT_COMPAT | ENT_HTML5);
echo $str; // "Волга", "Москвич"
```

Увидеть таблицу замен позволяет функция `get_html_translation_table()`:

```
print_r(
   get_html_translation_table(
      HTML_SPECIALCHARS, ENT_COMPAT | ENT_HTML5)
);
```

- `htmlentities()` — позволяет заменить все возможные специальные символы их HTML-эквивалентами:

```
$str = '"&()';
$str2 = htmlspecialchars($str, ENT_COMPAT | ENT_HTML5,
                        'UTF-8');
echo $str2; // &quot;&amp;()
$str = htmlentities($str, ENT_COMPAT | ENT_HTML5,
                   'UTF-8');
echo $str; // &quot;&amp;&lpar;&rpar;
```

С помощью функции `html_entity_decode()` можно выполнить обратное преобразование:

```
$str = '&quot;&amp;&lpar;&rpar;';
$str = html_entity_decode($str, ENT_COMPAT | ENT_HTML5, 'UTF-8');
echo $str; // "&()
```

Увидеть таблицу замен позволяет функция `get_html_translation_table()`:

```
print_r(
   get_html_translation_table(
      HTML_ENTITIES, ENT_COMPAT | ENT_HTML5)
);
```

- `wordwrap()` — позволяет разбить длинный текст на строки длины `$width`. Формат функции:

  ```
  wordwrap(string $str, int $width=75[, string $break="\n"[,
           bool $cut=false]]) : string
  ```

  Если в параметре `$cut` указано значение `true`, то значение параметра `$width` задает фиксированную длину в байтах. В этом случае функция будет корректно работать только с однобайтовыми кодировками:

  ```
  $str = "Очень длинная строка перед выводом";
  echo wordwrap($str, 7, "<br>");
  // Очень<br>длинная<br>строка<br>перед<br>выводом
  ```

- `nl2br()` — добавляет перед всеми символами новой строки (`\n`) тег `<br />`:

  ```
  $str = "Очень\nдлинная\nстрока\nперед\nвыводом";
  echo nl2br($str);
  ```

  Исходный HTML-код будет выглядеть следующим образом:

  ```
  Очень<br />
  длинная<br />
  строка<br />
  перед<br />
  выводом
  ```

  Если во втором параметре указать значение `false`, то будет добавляться тег `<br>`:

  ```
  $str = "Строка1\nСтрока2";
  echo nl2br($str, false);
  ```

  Исходный HTML-код будет выглядеть следующим образом:

  ```
  Строка1<br>
  Строка2
  ```

5.7.16. Преобразование строки в массив и обратно

Преобразовать строку в массив и обратно позволяют следующие функции:

- `explode()` — разделяет строку `$string` на подстроки по указанному разделителю `$delimiter` и добавляет полученные подстроки в массив. Формат функции:

  ```
  explode(string $delimiter, string $string[, int $limit]) : array
  ```

  Пример:

  ```
  $str = "A\tB\tC\tD";
  $arr = explode("\t", $str);
  print_r($arr);
  // Array ( [0] => A [1] => B [2] => C [3] => D )
  ```

  В необязательном параметре `$limit` можно указать максимальное количество элементов в итоговом массиве:

```
print_r( explode(";", "A;B;C;D", 2) );
// Array ( [0] => A [1] => B;C;D )
```

Если в параметре `$limit` задано отрицательное значение, то будут возвращены все подстроки, кроме последних, в количестве `-$limit`:

```
print_r( explode(";", "A;B;C;D", -2) );
// Array ( [0] => A [1] => B )
```

- `preg_split()` — разбивает строку по шаблону регулярного выражения и возвращает массив подстрок. Если не требуется указания шаблона, то вместо функции `preg_split()` лучше использовать функцию `explode()`:

```
$str = "A\tB\tC\t\t \nD";
$arr = preg_split('/\s+/u', $str);
print_r($arr);
// Array ( [0] => A [1] => B [2] => C [3] => D )
```

> ***Примечание***
>
> Регулярные выражения подробно рассматриваются в *разд. 5.8*.

- `implode()` — преобразует массив в строку. Элементы добавляются через указанный разделитель:

```
$arr = array(1, 2, 3, 4, 5);
$str = implode(', ', $arr);
echo $str; // 1, 2, 3, 4, 5
$str = implode(' - ', $arr);
echo $str; // 1 - 2 - 3 - 4 - 5
```

- `join()` — полностью аналогична функции `implode()`.

5.7.17. Кодирование и шифрование строк

Выполнить кодирование и шифрование строк позволяют следующие функции:

- `urlencode()` — выполняет URL-кодирование строки. Это необходимо, например, для передачи русского текста в строке URL-адреса в качестве параметра сценария:

```
$str = "Текст ";
echo urlencode($str);
// %D0%A2%D0%B5%D0%BA%D1%81%D1%82+
```

- `urldecode()` — раскодирует строку, закодированную с помощью функции `urlencode()`:

```
$str = "%D0%A2%D0%B5%D0%BA%D1%81%D1%82+";
echo "'", urldecode($str), "'"; // 'Текст '
```

Кроме этих функций, можно использовать функции `rawurlencode()` и `rawurldecode()`:

```
$str = "Текст ";
$str = rawurlencode($str);
echo $str;
// %D0%A2%D0%B5%D0%BA%D1%81%D1%82%20
echo "'", rawurldecode($str), "'"; // 'Текст '
```

> ***Внимание!***
>
> Символ пробела заменяется не знаком `+`, а символами `%20`.

- `mb_encode_mimeheader()` — позволяет закодировать текст с помощью методов `Base64` или `Quoted-Printable`. Формат функции:

  ```
  mb_encode_mimeheader(<Строка>[, <Кодировка>[,
            <Метод кодирования>[, <Символ перевода строк>[, <Отступ>]]]])
  ```

  Если параметр `<Кодировка>` не указан, то берется значение, указанное в функции `mb_internal_encoding()`. Как показывает практика, указывать кодировку в функции `mb_internal_encoding()` нужно обязательно. Параметр `<Метод кодирования>` может принимать значения `B` (`Base64`) или `Q` (`Quoted-Printable`). Если параметр не указан, то используется значение `B`. Параметр `<Символ перевода строк>` задает символ для разделения строк. По умолчанию предполагается комбинация `\r\n`.

  Функция `mb_decode_mimeheader(<Строка>)` позволяет выполнить обратную операцию.

  Пример:

  ```
  mb_internal_encoding('UTF-8');
  $theme = 'Сообщение';
  $theme = mb_encode_mimeheader($theme);
  echo $theme;
  // Выведет: =?UTF-8?B?0KHQvtC+0LHRidC10L3QuNC1?=
  echo mb_decode_mimeheader($theme);
  // Выведет: Сообщение
  ```

  Можно также воспользоваться функцией `iconv_mime_encode()` для кодирования строки и `iconv_mime_decode()` — для раскодирования:

  ```
  $arr = array(
     'scheme' => 'B',
     'input-charset' => 'UTF-8',
     'output-charset' => 'UTF-8',
     'line-length' => 76,
     'line-break-chars' => "\r\n"
  );
  $theme = 'Сообщение';
  $theme = iconv_mime_encode('Subject', $theme, $arr);
  echo $theme;
  // Subject: =?UTF-8?B?0KHQvtC+0LHRidC10L3QuNC1?=
  echo iconv_mime_decode($theme, 0, 'UTF-8');
  // Subject: Сообщение
  ```

- `md5()` — кодирует строку по алгоритму MD5. Часто используется для шифрования паролей. Для сравнения введенного пользователем пароля с сохраненным в базе необходимо зашифровать введенный пароль, а затем выполнить сравнение:

```
$passw = 'password';
$hash = md5($passw); // Пароль, сохраненный в базе
echo $hash; // Выведет: 5f4dcc3b5aa765d61d8327deb882cf99
$passw2 = 'password'; // Пароль, введенный пользователем
if ($hash === md5($passw2)) echo 'Пароль правильный';
```

- `sha1()` — шифрует строку:

```
$passw = 'password';
$hash = sha1($passw); // Пароль, сохраненный в базе
echo $hash; // 5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8
$passw2 = 'password'; // Пароль, введенный пользователем
if ($hash === sha1($passw2)) echo 'Пароль правильный';
```

- `password_hash()` — шифрует пароль. Формат функции:

```
password_hash(string $password, integer $algo[,
              array $options]) : string
```

  В первом параметре указывается пароль, во втором — алгоритм шифрования (`PASSWORD_DEFAULT` или `PASSWORD_BCRYPT`), а в третьем — массив с дополнительными параметрами.

  Для сравнения паролей используется функция `password_verify()`. Формат функции:

```
password_verify(string $password, string $hash) : boolean
```

  Пример:

```
$passw = 'password';
// Пароль, сохраненный в базе
$hash = password_hash($passw, PASSWORD_DEFAULT);
$passw2 = 'password'; // Пароль, введенный пользователем
if (password_verify($passw2, $hash))
   echo 'Пароль правильный';
```

5.7.18. Форматирование строки

Для форматированного вывода предназначена функция `printf()`. Формат функции:

```
printf(string $format[, mixed $args[, mixed $...]]) : int
```

В параметре `$format` указывается строка специального формата, внутри которой с помощью спецификаторов задаются правила форматирования. В параметре `$args` через запятую указываются различные значения. Функция возвращает длину выводимой строки:

```
printf('%.2f', 10.5125484);          // 10.51
```

Если мы не хотим сразу выводить результат, а собираемся сохранить его в виде строки, то следует воспользоваться функцией `sprintf()`. Формат функции:

```
sprintf(string $format[, mixed $args[, mixed $...]]) : string
```

Пример:

```
$str = sprintf('%.2f', 10.5125484);
echo $str; // 10.51
```

Обратите внимание: при работе функций `printf()` и `sprintf()` с многобайтовыми кодировками при указании максимальной ширины области можно получить неожиданный результат. Кроме того, результат выполнения зависит от локали. Если локаль не указана, то используется локаль по умолчанию. От настроек локали, например, зависит отображение десятичного разделителя вещественных чисел:

```
setlocale(LC_ALL, 'ru_RU', 'Russian_Russia');
printf('%.2f', 10.5125484);           // 10,51
```

Для форматирования данных, содержащихся в массиве, предназначены функции `vprintf()` и `vsprintf()`. Форматы функций:

```
vprintf(string $format, array $args) : int
vsprintf(string $format, array $args) : string
```

Пример:

```
vprintf("%.2f\n", [ 10.5125484 ]); // 10.51
$str = vsprintf("%.2f\n", [ 10.5125484 ]);
echo $str; // 10.51
```

В параметре `$format` задается строка специального формата, внутри которой могут быть указаны спецификаторы, имеющие следующий формат:

```
%[<Индекс>][<Флаги>][<Ширина>][.<Точность>]<Тип преобразования>
```

В параметре `$format` можно вообще не указывать спецификаторы. В этом случае значения в параметре `$args` не задаются:

```
printf('Просто строка'); // Просто строка
```

Если используется только один спецификатор, то параметр `$args` может содержать одно значение, в противном случае необходимо привести значения через запятую. Если количество значений не совпадает с количеством спецификаторов, то будет выведено сообщение об ошибке.

Вот пример указания двух спецификаторов и двух значений:

```
printf('%d %s', 10, 'руб.'); // 10 руб.
```

В этом примере строка содержит сразу два спецификатора: `%d` и `%s`. Спецификатор `%d` предназначен для вывода целого числа, а спецификатор `%s` — для вывода строки. Вместо спецификатора `%d` будет подставлено число `10`, а вместо спецификатора `%s` — строка `руб`. Обратите внимание: тип данных переданных значений должен совпадать с типом спецификатора. Если в качестве значения для спецификатора `%d` указать строку, то будет выполнено преобразование типов, и мы получим число `0`.

По умолчанию первому спецификатору соответствует первое значение, второму спецификатору — второе значение и т. д. Параметр `<Индекс>` позволяет изменить этот порядок и указать индекс значения. Обратите внимание на то, что индексы нумеруются с `1`, а не с нуля. После индекса указывается символ `$`:

```
printf('%2$s %1$d руб.', 10, 'цена'); // цена 10 руб.
```

Кроме того, один индекс можно указать несколько раз:

```
printf('%1$d %1$d', 10); // 10 10
```

Обратите внимание: символ `$` внутри двойных кавычек является специальным. Лучше использовать апострофы, в противном случае нужно экранировать этот символ обратным слешем:

```
printf("%1\$d %1\$d", 10); // 10 10
```

В параметре `<Тип преобразования>` могут быть указаны следующие символы:

- `c` — указанное число трактуется как ASCII-код символа:

  ```
  printf('%c', 81);          // Q
  ```

- `s` — строка (или любой другой тип, который автоматически будет преобразован в строку):

  ```
  printf('%s %s %s', 'строка', 10, 5.33);
  // строка 10 5.33
  ```

- `b` — двоичное число:

  ```
  printf('%b', 119);          // 1110111
  ```

- `d` — десятичное целое число со знаком:

  ```
  printf('%d %d', 10, -5);    // 10 -5
  ```

- `o` — восьмеричное число:

  ```
  printf('%o %o', 10, 077);   // 12 77
  ```

- `x` — шестнадцатеричное число в нижнем регистре:

  ```
  printf('%x %x', 10, 255);   // a ff
  ```

- `X` — шестнадцатеричное число в верхнем регистре:

  ```
  printf('%X %X', 10, 255);   // A FF
  ```

- `f` — вещественное число в десятичном представлении:

  ```
  printf("%f %f\n", 18.65781452, 12.5);
  // 18.657815 12.500000
  printf("%f\n", -18.65781452);       // -18.657815
  printf('%.0f %.2f', 100.0, 100.0); // 100 100.00
  ```

- `F` — вещественное число в десятичном представлении (не зависит от локали):

  ```
  printf("%F %F\n", 18.65781452, 12.5);
  // 18.657815 12.500000
  ```

```
printf("%F\n", -18.65781452);      // -18.657815
printf('%.0F %.2F', 100.0, 100.0); // 100 100.00
```

- e — вещественное число в экспоненциальной форме (буква e в нижнем регистре):

```
printf("%e\n", 18657.81452);   // 1.865781e+4
printf('%e', 0.000081452);     // 8.145200e-5
```

- E — вещественное число в экспоненциальной форме (буква e в верхнем регистре):

```
printf('%E', 18657.81452);     // 1.865781E+4
```

- g — эквивалентно f или e (выбирается более короткая запись числа):

```
printf('%g %g %g', 0.086578, 0.000086578, 1.865E-005);
// 0.086578 8.6578e-5 1.865e-5
```

- G — эквивалентно f или E (выбирается более короткая запись числа):

```
printf('%G %G %G', 0.086578, 0.000086578, 1.865E-005);
// 0.086578 8.6578E-5 1.865E-5
```

- % — символ процента (%):

```
printf('10%%'); // 10%
```

Параметр <Ширина> задает минимальную ширину поля. Если строка меньше ширины поля, то она дополняется пробелами. Если строка не помещается в указанную ширину, то значение игнорируется и строка выводится полностью:

```
printf("'%3s'", 'string');  // 'string'
printf("'%10s'", 'string'); // '    string'
```

Задать минимальную ширину можно не только для строк, но и для других типов:

```
printf("'%10d'", 25);   // '        25'
printf("'%10f'", 12.5); // ' 12.500000'
```

По умолчанию выравнивание производится по правой границе области. Если перед шириной указать флаг -, то будет выполнено выравнивание по левой границе области:

```
printf("'%10s'", 'string');  // '    string'
printf("'%-10s'", 'string'); // 'string    '
```

Параметр <Точность> задает количество знаков после точки для вещественных чисел. Перед этим параметром обязательно должна стоять точка:

```
printf("'%10.5f'", 3.14159265359); // '   3.14159'
printf("'%.3f'", 3.14159265359);   // '3.142'
```

Если параметр <Точность> используется применительно к строке, то он задает максимальное количество байтов. Символы, которые не помещаются, будут отброшены:

```
printf("'%5.7s'", "Hello, world!");   // 'Hello, '
printf("'%15.20s'", "Hello, world!"); // '  Hello, world!'
```

ВНИМАНИЕ!

При работе с многобайтовыми кодировками будут проблемы, т. к. параметр `<Точность>` задает число байтов, а не символов.

В параметре `<Флаги>` могут быть указаны следующие символы или их комбинация:

- `0` — задает наличие ведущих нулей для числового значения:

```
printf("'%7d'", 100);          // '    100'
printf("'%07d'", 100);         // '0000100'
```

- `-` — задает выравнивание по левой границе области. По умолчанию используется выравнивание по правой границе:

```
printf("'%5d' '%-5d'", 3, 3); // '    3' '3    '
printf("'%05d'", 3);          // '00003'
```

- `+` — обязательный вывод знака, как для отрицательных, так и для положительных чисел:

```
printf("'%+d' '%+d'", -3, 3); // '-3' '+3'
```

- `'символ` — задает символ-заполнитель. По умолчанию используется пробел:

```
printf("'%7d'\n", 100);        // '    100'
printf("'%'.7d'\n", 100);      // '....100'
printf("'%'_7d'\n", 100);      // '____100'
printf("'%'07d'", 100);        // '0000100'
```

Все результаты, которые были показаны в приведенных примерах, справедливы для следующих настроек локали:

```
echo setlocale(LC_ALL, '0');
// LC_COLLATE=C;LC_CTYPE=Russian_Russia.1251;
// LC_MONETARY=C;LC_NUMERIC=C;LC_TIME=C
```

Если изменить локаль, то и результат будет соответствовать настройкам указанной локали. В частности, от локали зависит символ десятичного разделителя:

```
printf("%.2f\n", 10.5125484);          // 10.51
setlocale(LC_ALL, 'ru_RU', 'Russian_Russia');
printf('%.2f', 10.5125484);            // 10,51
```

Для форматирования чисел можно также воспользоваться функцией `number_format()`, которую мы уже рассмотрели в *разд. 5.5.6*:

```
$x = 1234567.126;
echo number_format($x), "\n";          // 1,234,567
echo number_format($x, 2), "\n";       // 1,234,567.13
echo number_format($x, 2, ',', ' '); // 1 234 567,13
```

5.8. Регулярные выражения PCRE

Регулярные выражения позволяют осуществить в строке сложный поиск или замену. В языке PHP существуют два формата регулярных выражений: POSIX и PCRE (Perl-compatible Regular Expression, Perl-совместимые регулярные выражения). Синтаксис их схож, но скорость и внутренний механизм работы сильно различаются. В PHP 7 функции, которые позволяют использовать регулярные выражения формата POSIX, удалены. Тем не менее функции с префиксом `mb_ereg` все еще позволяют использовать формат POSIX, но мы их рассматривать не будем, т. к. для работы с кодировкой UTF-8, а также с однобайтовыми кодировками лучше использовать PCRE.

5.8.1. Создание шаблона

Шаблон PCRE представляет собой строку, заключенную в кавычки или апострофы, внутри которой между двумя ограничителями указывается регулярное выражение. За последним ограничителем могут быть указаны модификаторы. В качестве ограничителя могут выступать одинаковые символы или парные скобки:

```
'/<Регулярное выражение>/[<Модификаторы>]'
'#<Регулярное выражение>#[<Модификаторы>]'
'"<Регулярное выражение>"[<Модификаторы>]'
'{<Регулярное выражение>}[<Модификаторы>]'
'(<Регулярное выражение>)[<Модификаторы>]'
```

В параметре `<Модификаторы>` могут быть указаны следующие флаги (или их комбинация):

- `i` — поиск без учета регистра:

  ```
  var_dump( preg_match('/абв/u', 'аБв') );  // int(0)
  var_dump( preg_match('/абв/iu', 'аБв') ); // int(1)
  ```

- `m` — поиск в строке, состоящей из нескольких строк, разделенных символом новой строки. Символ `^` соответствует привязке к началу каждой подстроки, а символ `$` соответствует позиции перед символом перевода строки. Метасимвол «точка» соответствует любому символу, кроме символа перевода строки (`\n`);
- `s` — метасимвол «точка» соответствует любому символу, в том числе и символу перевода строки:

  ```
  var_dump( preg_match('/^.+$/u', "а\nв") );  // int(0)
  var_dump( preg_match('/^.+$/su', "а\nв") ); // int(1)
  ```

- `x` — разрешает использовать в регулярном выражении пробельные символы и однострочные комментарии, начинающиеся с символа `#`;
- `U` — ограничивает «жадность» всех квантификаторов в шаблоне. Обратите внимание на регистр модификатора — буква должна быть прописной;
- `u` — служит для обработки строк в кодировке UTF-8. Регистр модификатора `u` имеет значение. Так как мы работаем с кодировкой UTF-8, то этот модификатор

обязательно должен присутствовать. В качестве примера удалим из строки все русские буквы:

```
$str = 'строка1строка2Строка3';
echo preg_replace('/[а-яё]/iu', '', $str); // 123
```

Если в этом примере модификатор `u` не указать, то будет удален один байт из каждого двухбайтового символа и в итоге в строке появятся квадратики или знаки вопроса.

> **ПРИМЕЧАНИЕ**
>
> Существует также другие модификаторы, которые используются очень редко. Полный их список смотрите в документации.

Модификаторы можно установить или сбросить внутри шаблона регулярного выражения с помощью следующих конструкций:

- `(?imsxUXJ-imsxUXJ)` — позволяет установить или сбросить (если буква указана после знака `-`) модификаторы внутри регулярного выражения. Буквы соответствуют модификаторам в шаблоне.

  Вот пример установки модификаторов:

  ```
  var_dump( preg_match('/^[а-яё]+$/u', 'АБВЁ') );      // int(0)
  var_dump( preg_match('/(?i)^[а-яё]+$/u', 'АБВЁ') ); // int(1)
  var_dump( preg_match('/^.+$/u', "а\nв") );          // int(0)
  var_dump( preg_match('/(?s)^.+$/u', "а\nв") );      // int(1)
  ```

- `(?imsxUXJ-imsxUXJ:шаблон)` — позволяет установить или сбросить (если буква указана после знака `-`) модификаторы внутри регулярного выражения. В этом случае шаблон регулярного выражения располагается внутри круглых скобок после двоеточия, что позволяет установить или сбросить модификаторы только для фрагмента шаблона.

  Вот пример установки модификаторов:

  ```
  var_dump( preg_match('/^[a-z]+ [a-z]+$/u', 'ABC DEF') );
  // int(0)
  var_dump( preg_match('/(?i:^[a-z]+) [a-z]+$/u', 'ABC DEF') );
  // int(0)
  var_dump(preg_match('/(?i:^[a-z]+) [a-zA-Z]+$/u', 'ABC DEF'));
  // int(1)
  var_dump(
      preg_match('/(?i:^[a-z]+) (?i:[a-z]+$)/u', 'ABC DEF') );
  // int(1)
  ```

5.8.2. Синтаксис регулярных выражений

Обычные символы, не имеющие специального значения, могут присутствовать в строке шаблона, и они будут трактоваться как есть. Вот пример указания в шаблоне последовательности обычных символов:

```
var_dump( preg_match('/строка/u', 'строка') ); // int(1)
```

Экранирование специальных символов

Внутри регулярного выражения символы ., ^, *, +, ?, {, [,], \, |, (и), а также закрывающий символ шаблона, имеют специальное значение. Если эти символы должны трактоваться как есть, то их следует экранировать с помощью слеша. Некоторые специальные символы теряют свое специальное значение, если их разместить внутри квадратных скобок. В этом случае экранировать их не нужно. Например, по умолчанию метасимвол «точка» соответствует любому символу, кроме символа перевода строки. Если необходимо найти именно точку, то перед ней нужно указать символ \ или разместить точку внутри квадратных скобок: [.]. Продемонстрируем это на примере проверки правильности введенной даты (листинг 5.22.)

Листинг 5.22. Проверка правильности ввода даты

```
<?php
$str = '29,10.2017'; // Неправильная дата (вместо точки указана запятая)

$pattern = '/^[0-3][0-9].[01][0-9].[12][09][0-9][0-9]$/su';
// Символ \ не указан перед точкой
if (preg_match($pattern, $str)) echo 'Дата введена правильно';
else echo 'Дата введена неправильно';
// Так как точка означает любой символ, выведет: Дата введена правильно

$pattern = '/^[0-3][0-9]\.[01][0-9]\.[12][09][0-9][0-9]$/su';
// Символ \ указан перед точкой
if (preg_match($pattern, $str)) echo 'Дата введена правильно';
else echo 'Дата введена неправильно';
// Так как перед точкой указан символ \,
// выведет: Дата введена неправильно

$pattern = '/^[0-3][0-9][.][01][0-9][.][12][09][0-9][0-9]$/su';
// Точка внутри квадратных скобок
if (preg_match($pattern, $str)) echo 'Дата введена правильно';
else echo 'Дата введена неправильно';
// Выведет: Дата введена неправильно
```

Следует учитывать, что символ обратного слеша является специальным не только в шаблоне регулярного выражения, но и в строке. Если метасимвола нет, то можно указать только один символ обратного слеша:

```
var_dump( preg_match('/\ /u', '\ ') );    // int(1)
var_dump('/\ /u');  // string(5) "/\ /u"
var_dump('\ ');     // string(2) "\ "
```

Однако если слеш расположен в самом конце строки, или за слешем идет метасимвол или закрывающий символ шаблона, то слеш нужно дополнительно экранировать:

```
var_dump( preg_match('/\\\\/u', '\\') );    // int(1)
var_dump('/\\\\/u');  // string(5) "/\\/u"
var_dump('\\');      // string(1) "\"
```

Посмотрите, сколько слешей добавилось! Чтобы не путаться, следует выводить строку с шаблоном на экран и смотреть, что получилось в итоге. Помните, что экранирование внутри шаблона должно быть видно при выводе.

Давайте рассмотрим еще один пример. Нужно проверить, равна ли строка значению `'\s'`. Внутри апострофов большинство специальных символов не преобразуются, поэтому в самой строке можно указать только один слеш, хотя лучше указать два: `'\\s'`. Внутри шаблона регулярного выражения эта комбинация символов является специальной и обозначает любые пробельные символы. Поэтому внутри шаблона мы должны добавить еще три слеша, чтобы экранировать специальную комбинацию:

```
var_dump( preg_match('/^\\\\s$/su', '\\s') );    // int(1)
var_dump('/^\\\\s$/su');  // string(9) "/^\\s$/su"
var_dump('\\s');          // string(2) "\s"
```

Использовать для шаблонов строки в двойных кавычках вообще не рекомендуется, т. к. в этом случае нам пришлось бы еще дополнительно экранировать символ `$`.

Экранировать все специальные символы в строке позволяет функция `preg_quote()`. Формат функции:

```
preg_quote(string $str[, string $delimiter]) : string
```

В необязательном параметре `$delimiter` можно указать дополнительный символ, требующий экранирования, — например, закрывающий символ шаблона:

```
$str = '. ^ * + - ? { [ ] \ | ( ) $ < > ! | / ';
echo preg_quote($str, '/');
// \. \^ \* \+ \- \? \{ \[ \] \\ \| \( \) \$ \< \> \! \| \/
```

Метасимволы

Приведем метасимволы, применяемые в регулярных выражениях:

- `^` — привязка к началу строки (назначение зависит от модификатора `m`);
- `$` — привязка к концу строки (назначение зависит от модификатора `m`);
- `\A` — привязка к началу строки (не зависит от модификатора);
- `\z` — привязка к концу строки (не зависит от модификатора);
- `[]` — позволяет указать символы, которые могут встречаться на этом месте в строке. Можно записать символы подряд или указать диапазон через тире;
- `[^]` — позволяет указать символы, которые не могут встречаться на этом месте в строке. Можно записать символы подряд или указать диапазон через тире;
- `n|m` — соответствует одному из символов `n` или `m`;

- . (точка) — любой символ, кроме символа перевода строки (`\n`). Если указан модификатор `s`, то метасимвол «точка» соответствует всем символам, включая символ перевода строки. Внутри квадратных скобок точка не имеет специального значения.

Привязку к началу и концу строки следует использовать, если строка должна полностью соответствовать регулярному выражению. Например, проверим, содержит ли строка только число:

```
$pattern = '/^[0-9]+$/u';
$str = '2';
if (preg_match($pattern, $str)) echo 'Число'; // Выведет: Число
else echo 'Не число';
$str = 'Строка2';
if (preg_match($pattern, $str)) echo 'Число';
else echo 'Не число'; // Выведет: Не число
```

Если привязку не указать, то любая строка, содержащая число, пройдет проверку:

```
$pattern = '/[0-9]+/u';
$str = 'Строка2';
if (preg_match($pattern, $str)) echo 'Число'; // Выведет: Число
else echo 'Не число';
```

Можно указать привязку только к началу или только к концу строки:

```
$pattern = '/[0-9]+$/u';
$str = 'Строка2';
if (preg_match($pattern, $str)) echo 'Есть число в конце строки';
else echo 'Нет числа в конце строки';
// Выведет: Есть число в конце строки
$pattern = '/^[0-9]+/u';
if (preg_match($pattern, $str)) echo 'Есть число в начале строки';
else echo 'Нет числа в начале строки';
// Выведет: Нет числа в начале строки
```

По умолчанию используется однострочный режим. Например, не указав привязку к началу и концу, в этом примере мы получим все три числа:

```
$pattern = '/[0-9]+/u';
$str = "10\n20\n30";
$arr = array();
preg_match_all($pattern, $str, $arr);
print_r($arr);
// Array ( [0] => Array ( [0] => 10 [1] => 20 [2] => 30 ) )
```

Если мы первую инструкцию изменим следующим образом:

```
$pattern = '/^[0-9]+$/u';
```

то не будет найдено ни одного соответствия, т. к. указана привязка к концу и началу строки. Однако если указан модификатор `m`, то поиск производится в строке,

состоящей из нескольких подстрок, разделенных символом новой строки. В этом случае символ `^` соответствует привязке к началу каждой подстроки, а символ `$` соответствует позиции перед символом перевода строки:

```
$pattern = '/^[0-9]+$/mu';
$str = "10\n20\n30";
$arr = array();
preg_match_all($pattern, $str, $arr);
print_r($arr);
// Array ( [0] => Array ( [0] => 10 [1] => 20 [2] => 30 ) )
```

Хотя мы и указали привязку, все равно получим все числа, т. к. привязка действует к началу и концу каждой отдельной подстроки.

Чтобы в режиме m работала привязка к началу и концу всей строки, необходимо использовать метасимволы `\A` и `\z`:

```
$pattern = '/\\A[0-9]+\\z/mu';
```

В квадратных скобках `[]` можно указать символы, которые могут встречаться на этом месте в строке. Можно записать символы подряд или указать диапазон через тире:

- `[09]` — соответствует числу `0` или `9`;
- `[0-9]` — соответствует любому числу от `0` до `9`;
- `[абв]` — соответствует буквам `а`, `б` и `в`;
- `[а-г]` — соответствует буквам `а`, `б`, `в` и `г`;
- `[а-яё]` — соответствует любой букве от `а` до `я`;
- `[АБВ]` — соответствует буквам `А`, `Б` и `В`;
- `[А-ЯЁ]` — соответствует любой букве от `А` до `Я`;
- `[а-яА-ЯёЁ]` — соответствует любой русской букве в любом регистре;
- `[0-9а-яА-ЯёЁa-zA-Z]` — любая цифра и любая русская или английская буква независимо от регистра.

> ***Внимание!***
> Буква `ё` не входит в диапазон `[а-я]`.

Значение можно инвертировать, если после первой скобки указать символ `^`. Таким способом можно указать символы, которых не должно быть на этом месте в строке:

- `[^09]` — не цифра `0` или `9`;
- `[^0-9]` — не цифра от `0` до `9`;
- `[^а-яА-ЯёЁa-zA-Z]` — не русская или английская буква в любом регистре.

Как вы уже знаете, точка теряет свое специальное значение, если ее заключить в квадратные скобки. Кроме того, внутри квадратных скобок могут встретиться символы, которые имеют специальное значение (например, `^` и `-`). Символ `^` теряет

свое специальное значение, если он не расположен сразу после открывающей квадратной скобки:

```
$pattern = '/[09^]/u';   // 0, 9 или ^
```

Чтобы отменить специальное значение символа -, его необходимо указать после всех символов перед закрывающей квадратной скобкой:

```
$pattern = '/[09-]/u';   // 0, 9 или -
```

Все специальные символы можно сделать обычными, если перед ними указать символ \:

```
$pattern = '/[0\\-9]/u'; // 0, - или 9
```

Стандартные классы

В регулярных выражениях допустимы следующие основные стандартные классы (полный их список смотрите в документации):

- \d — соответствует любой цифре;
- \w — соответствует любой букве или цифре;
- \s — любой пробельный символ (пробел, табуляция, перевод страницы, новая строка или перевод каретки);
- \D — не цифра. Эквивалентно [^\d];
- \W — не буква и не цифра. Эквивалентно [^\w];
- \S — не пробельный символ. Эквивалентно [^\s].

Получим все цифры из строки:

```
$pattern = '/\\d/u';
$str = "строка123";
$arr = array();
preg_match_all($pattern, $str, $arr);
print_r($arr);
// Array ( [0] => Array ( [0] => 1 [1] => 2 [2] => 3 ) )
```

Если нужно получить не цифры по отдельности, а числа, то после класса надо указать символ +, означающий одно или большее число вхождений символа в строку:

```
$pattern = '/\\d+/u';
$str = "строка123";
$arr = array();
preg_match_all($pattern, $str, $arr);
print_r($arr);
// Array ( [0] => Array ( [0] => 123 ) )
```

Следует учитывать, что стандартные классы при использовании кодировки UTF-8 могут трактоваться гораздо шире, чем вы можете подумать. В итоге можно получить результат, который совсем не ожидался. Советуем на практике использовать только классы \s и \S, а остальные заменять явным указанием диапазона символов внутри квадратных скобок (при этом не забывайте про букву ё):

```
$pattern = '/[0-9]/u';          // Вместо \d
$pattern = '/[0-9а-яёa-z]/u'; // Вместо \w
```

Внутри квадратных скобок можно также использовать стандартные классы регулярных выражений формата POSIX:

- `[[:alnum:]]` — алфавитно-цифровые символы;
- `[[:word:]]` — аналог класса `\w`;
- `[[:alpha:]]` — буквенные символы;
- `[[:lower:]]` — строчные буквы;
- `[[:upper:]]` — прописные буквы;
- `[[:digit:]]` — десятичные цифры;
- `[[:xdigit:]]` — шестнадцатеричные цифры;
- `[[:punct:]]` — знаки пунктуации;
- `[[:blank:]]` — символы табуляции и пробелов;
- `[[:space:]]` — пробельные символы;
- `[[:cntrl:]]` — управляющие символы;
- `[[:print:]]` — печатные символы;
- `[[:graph:]]` — печатные символы, за исключением пробела;
- `[[:ascii:]]` — символы с кодами от 0 до 127.

Получим все числа из строки:

```
$pattern = '/[[:digit:]]+/u';
$str = "123 456 789";
$arr = array();
preg_match_all($pattern, $str, $arr);
print_r($arr);
// Array ( [0] => Array ( [0] => 123 [1] => 456 [2] => 789 ) )
```

Квантификаторы

Количество вхождений символа (или выражения) в строку задается с помощью *квантификаторов*:

- `{n}` — `n` вхождений символа в строку (шаблон `[0-9]{2}` соответствует двум вхождениям любой цифры);
- `{n,}` — `n` или более вхождений символа в строку (шаблон `[0-9]{2,}` соответствует двум и более вхождениям любой цифры);
- `{n,m}` — не менее `n` и не более `m` вхождений символа в строку. Числа указываются через запятую без пробела. Например, шаблон `[0-9]{2,4}` соответствует от двух до четырех вхождений любой цифры;
- `*` — ноль или большее число вхождений символа в строку. Эквивалентно комбинации `{0,}`;

- `+` — одно или большее число вхождений символа в строку. Эквивалентно комбинации `{1,}`;
- `?` — ни одного или одно вхождение символа в строку. Эквивалентно комбинации `{0,1}`.

«Жадность» квантификаторов

Все квантификаторы являются «жадными» — при поиске соответствия ищется самая длинная подстрока, соответствующая шаблону, и не учитываются более короткие соответствия. Рассмотрим это на примере, в котором получим содержимое всех тегов `<b>` вместе с самими тегами:

```
$pattern = '#<b>.*</b>#siu';
$str = '<b>Text1</b>Text2<b>Text3</b>';
$arr = array();
preg_match_all($pattern, $str, $arr);
print_r($arr);
// Array ( [0] => Array ( [0] => <b>Text1</b>Text2<b>Text3</b> ) )
```

Вместо желаемого результата мы получили полностью строку. Чтобы ограничить «жадность», необходимо после квантификатора указать символ `?`:

```
$pattern = '#<b>.*?</b>#siu';
$str = '<b>Text1</b>Text2<b>Text3</b>';
$arr = array();
preg_match_all($pattern, $str, $arr);
print_r($arr);
```

Этот код выведет то, что мы искали:

```
Array ( [0] => Array ( [0] => <b>Text1</b> [1] => <b>Text3</b> ) )
```

Ограничить «жадность» всех квантификаторов в шаблоне позволяет модификатор `U`. Обратите внимание на регистр модификатора — буква должна быть прописной:

```
$pattern = '#<b>.*</b>#siUu';
$pattern = '#(?U)<b>.*</b>#siu';
$pattern = '#<b>(?U:.*)</b>#siu';
```

Группы

Если необходимо получить содержимое без тегов, то нужный фрагмент внутри шаблона следует разместить внутри круглых скобок:

```
$pattern = '#<b>(.*?)</b>#siu';
$str = '<b>Text1</b>Text2<b>Text3</b>';
$arr = array();
preg_match_all($pattern, $str, $arr, PREG_PATTERN_ORDER);
```

Результат будет доступен через элемент массива с индексом, совпадающим с порядковым номером круглых скобок внутри шаблона (нумерация начинается с единицы). Указав индекс `1`, мы получим фрагмент, соответствующий `(.*?)`, т. е. текст внутри тегов `<b>`:

```
print_r($arr[1] ?? "Нет элементов");
// Array ( [0] => Text1 [1] => Text3 )
```

Элемент массива с индексом `0` будет содержать фрагменты, полностью совпадающие с шаблоном:

```
print_r($arr[0] ?? "Нет элементов");
// Array ( [0] => <b>Text1</b> [1] => <b>Text3</b> )
```

Круглые скобки часто используются для группировки фрагментов внутри шаблона. В этих случаях не требуется, чтобы фрагмент запоминался и был доступен в результатах поиска:

```
$pattern = '#([a-z]+((st)|(xt)))#siu';
$str = 'test text';
$arr = array();
preg_match_all($pattern, $str, $arr, PREG_PATTERN_ORDER);
print_r($arr);
// Array (
// [0] => Array ( [0] => test [1] => text )
// [1] => Array ( [0] => test [1] => text )
// [2] => Array ( [0] => st [1] => xt )
// [3] => Array ( [0] => st [1] => )
// [4] => Array ( [0] => [1] => xt ) )
```

В этом примере мы получили массив из элементов для каждого совпадения. Все эти элементы (кроме элемента с индексом `0`) соответствуют фрагментам, заключенным в шаблоне в круглые скобки. Элемент с индексом `1` содержит фрагменты, расположенные в первых круглых скобках, с индексом `2` — во вторых круглых скобках и т. д. Три последних элемента являются лишними. Чтобы избежать захвата фрагмента, после открывающих круглых скобок следует разместить символы `?:`:

```
$pattern = '#([a-z]+(?:(?:st)|(?:xt)))#siu';
$str = 'test text';
$arr = array();
preg_match_all($pattern, $str, $arr, PREG_PATTERN_ORDER);
print_r($arr);
// Array (
// [0] => Array ( [0] => test [1] => text )
// [1] => Array ( [0] => test [1] => text ) )
```

В результате массив состоит только из фрагментов, полностью соответствующих регулярному выражению.

Обратите внимание на регулярное выражение в предыдущем примере:

```
$pattern = '#([a-z]+((st)|(xt)))#siu'
```

Здесь мы использовали метасимвол `|`, который позволяет сделать выбор между альтернативными значениями. Выражение `n|m` соответствует одному из символов: `n` или `m`:

- `красн((ая)|(ое))` — `красная` или `красное`, но не красный.

Обратные ссылки

К найденному фрагменту в круглых скобках внутри шаблона можно обратиться с помощью механизма обратных ссылок. Для этого порядковый номер круглых скобок в шаблоне указывается после слеша, например `\1`. Нумерация скобок внутри шаблона начинается с 1. Альтернативные варианты обращения: `\g1` и `\g{1}`.

Для примера получим текст между одинаковыми парными тегами:

```
$pattern = '#<([a-z]+)[^>]*?>(.*?)</\\1>#isu';
$str = '<b>Text1</b>Text2<I>Text3</I>';
$arr = array();
preg_match_all($pattern, $str, $arr, PREG_SET_ORDER);
print_r($arr);
// Array (
// [0] => Array ( [0] => <b>Text1</b> [1] => b [2] => Text1 )
// [1] => Array ( [0] => <I>Text3</I> [1] => I [2] => Text3 ) )
```

Фрагментам внутри круглых скобок можно дать имена. Для этого следует указать одну из следующих комбинаций символов:

```
(?P<имя>шаблон)
(?<имя>шаблон)
(?'имя'шаблон)
```

В качестве примера разберем E-mail на составные части:

```
$pattern =
   '#(?<name>[a-z0-9_.-]+)@(?<host>(?:[a-z0-9-]+\\.)+[a-z]{2,6})#siu';
$str = 'user@mail.ru';
$arr = array();
preg_match_all($pattern, $str, $arr, PREG_SET_ORDER);
print_r($arr);
// Array (
// [0] => Array (
//          [0] => user@mail.ru
//          [name] => user
//          [1] => user
//          [host] => mail.ru
//          [2] => mail.ru ) )
```

Чтобы внутри шаблона обратиться к именованным фрагментам, используется следующий синтаксис:

```
(?P=имя)
\k<имя>
\k{имя}
\k'имя'
\g<имя>
\g{имя}
\g'имя'
```

Для примера получим текст между одинаковыми парными тегами:

```
$pattern =
   '#<(?<tag>[a-z]+)>(?<text>.*?)</(\\k<tag>)>#siu';
$str = '<b>Text1</b>Text2<I>Text3</I>';
$arr = array();
preg_match_all($pattern, $str, $arr, PREG_SET_ORDER);
foreach ($arr as $v) {
   echo $v['text'] . "<br>\n";
}
```

Результат в окне Web-браузера:

```
Text1
Text3
```

Просмотр вперед или назад

Внутри круглых скобок могут быть расположены следующие конструкции:

- `(?=...)` — положительный просмотр вперед. Выведем все слова, после которых расположена запятая:

```
$pattern = '#\\w+(?=[,])#siu';
$str = 'text1, text2, text3 text4';
$arr = array();
preg_match_all($pattern, $str, $arr, PREG_SET_ORDER);
foreach ($arr as $v) {
   echo $v[0] . " ";
} // text1 text2
```

- `(?!...)` — отрицательный просмотр вперед. Выведем все слова, после которых нет запятой:

```
$pattern = '#[a-z]+[0-9](?![,])#siu';
$str = 'text1, text2, text3 text4';
$arr = array();
preg_match_all($pattern, $str, $arr, PREG_SET_ORDER);
foreach ($arr as $v) {
   echo $v[0] . " ";
} // text3 text4
```

- `(?<=...)` — положительный просмотр назад. Выведем все слова, перед которыми расположена запятая с пробелом:

```
$pattern = '#(?<=[,][ ])[a-z]+[0-9]#siu';
$str = 'text1, text2, text3 text4';
$arr = array();
preg_match_all($pattern, $str, $arr, PREG_SET_ORDER);
foreach ($arr as $v) {
   echo $v[0] . " ";
} // text2 text3
```

- `(?<!...)` — отрицательный просмотр назад. Выведем все слова, перед которыми расположен пробел, но перед пробелом нет запятой:

```
$pattern = '#(?<![,]) ([a-z]+[0-9])#siu';
$str = 'text1, text2, text3 text4';
$arr = array();
preg_match_all($pattern, $str, $arr, PREG_SET_ORDER);
foreach ($arr as $v) {
   echo $v[1] . " ";
} // text4
```

Рассмотрим небольшой пример. Предположим, необходимо получить все слова, расположенные после тире, причем перед тире и после слов должны следовать пробельные символы:

```
$pattern = '#\\s\\-([a-z0-9]+)\\s#siu';
$str = '-word1 -word2 -word3 -word4 -word5';
$arr = array();
preg_match_all($pattern, $str, $arr, PREG_SET_ORDER);
foreach ($arr as $v) {
   echo $v[1] . " ";
} // word2 word4
```

Как видно из результата, мы получили только два слова вместо пяти. Первое и последнее слова не попали в результат, т. к. расположены в начале и в конце строки. Чтобы эти слова попали в результат, необходимо добавить альтернативный выбор для начала строки: (`^|\\s`) и для конца строки: (`\\s|$`). Чтобы найденные выражения внутри круглых скобок не попали в результат, следует добавить после открывающих скобок символы `?:`:

```
$pattern = '#(?:^|\\s)\\-([a-z0-9]+)(?:\\s|$)#siu';
$str = '-word1 -word2 -word3 -word4 -word5';
$arr = array();
preg_match_all($pattern, $str, $arr, PREG_SET_ORDER);
foreach ($arr as $v) {
   echo $v[1] . " ";
} // word1 word3 word5
```

Первое и последнее слова успешно попали в результат. Почему же слова `word2` и `word4` не попали в список совпадений? Ведь перед тире есть пробел и после слова есть пробел. Чтобы понять причину, рассмотрим поиск по шагам. Первое слово успешно попадает в результат, т. к. перед тире расположено начало строки и после слова есть пробел. После поиска указатель перемещается, и строка для дальнейшего поиска примет следующий вид:

```
-word1 <Указатель>-word2 -word3 -word4 -word5
```

Обратите внимание на то, что перед фрагментом `-word2` больше нет пробела, и тире не расположено в начале строки. Поэтому следующим совпадением будет слово `word3`, и указатель снова будет перемещен:

```
-word1 -word2 -word3 <Указатель>-word4 -word5
```

Опять перед фрагментом `-word4` нет пробела, и тире не расположено в начале строки. Поэтому следующим совпадением окажется слово `word5`, и поиск будет завершен. Таким образом, слова `word2` и `word4` не попадают в результат, т. к. пробел до фрагмента уже был использован в предыдущем поиске. Чтобы этого избежать, следует воспользоваться положительным просмотром вперед `(?=...)`:

```
$pattern = '#(?:^|\\s)\\-([a-z0-9]+)(?=\\s|$)#siu';
$str = '-word1 -word2 -word3 -word4 -word5';
$arr = array();
preg_match_all($pattern, $str, $arr, PREG_SET_ORDER);
foreach ($arr as $v) {
   echo $v[1] . " ";
} // word1 word2 word3 word4 word5
```

В этом примере мы заменили фрагмент `(?:\\s|$)` на `(?=\\s|$)`. Поэтому все слова успешно попали в список совпадений.

5.8.3. Сравнение с шаблоном

Функция `preg_match()` выполняет сравнение с шаблоном. Формат функции:

```
preg_match(string $pattern, string $subject[, array &$matches[,
           int $flags=0[, int $offset=0]]]) : int
```

В первом параметре указывается шаблон регулярного выражения, а во втором — строка, с которой выполняется сравнение. Функция возвращает `0`, если совпадение не найдено, `1` — при соответствии шаблону и `false` — в случае ошибки.

Если указан параметр `$matches`, то первый элемент массива будет содержать фрагмент, полностью соответствующий шаблону, а остальные элементы массива — это фрагменты, заключенные в шаблоне в круглые скобки.

В качестве примера проверим E-mail на соответствие шаблону:

```
$email = 'user@mail.ru';
$pattern = '/^([a-z0-9_.-]+)@(([a-z0-9-]+\\.)+[a-z]{2,6})$/isu';
$arr = array();
if (preg_match($pattern, $email, $arr)) {
   echo 'E-mail ' . $arr[0] . ' соответствует шаблону';
   echo '<br>пользователь: ', $arr[1], ' домен: ', $arr[2];
}
else {
   echo 'E-mail не соответствует шаблону';
}
```

Результат в окне Web-браузера:

```
E-mail user@mail.ru соответствует шаблону
пользователь: user домен: mail.ru
```

Выведем структуру массива `$arr`:

```
print_r($arr);
// Array ( [0] => user@mail.ru [1] => user [2] => mail.ru [3] => mail.)
```

- `$arr[0]` — текст, полностью соответствующий шаблону;
- `$arr[1]` — соответствует группе `([a-z0-9_.-]+)`;
- `$arr[2]` — соответствует группе `(([a-z0-9-]+\.)+[a-z]{2,6})`;
- `$arr[3]` — соответствует группе `([a-z0-9-]+\.)`.

Если какой-либо фрагмент заносить в массив не надо, то после открывающей круглой скобки следует указать комбинацию символов `?:`. Последний элемент является лишним, давайте от него избавимся:

```
$pattern = '/^([a-z0-9_.-]+)@((?:[a-z0-9-]+\\.)+[a-z]{2,6})$/isu';
```

Результат:

```
Array ( [0] => user@mail.ru [1] => user [2] => mail.ru )
```

Если внутри шаблона находятся именованные группы, то массив будет дополнительно содержать элементы с именами групп:

```
$pattern =
  '#(?<name>[a-z0-9_.-]+)@(?<host>(?:[a-z0-9-]+\\.)+[a-z]{2,6})#isu';
```

Результат:

```
Array ( [0] => user@mail.ru [name] => user [1] => user
        [host] => mail.ru [2] => mail.ru )
```

Если в параметре `$flags` задано значение `PREG_OFFSET_CAPTURE`, то для каждого найденного фрагмента будет указана его позиция в исходной строке. Для примера получим текст между одинаковыми парными тегами и выведем смещение относительно начала строки:

```
$str = "<b>Текст</b>";
$pattern = '#<([a-z]+)[^>]*?>(.*?)</\\1>#isu';
$arr = array();
if (preg_match($pattern, $str, $arr, PREG_OFFSET_CAPTURE)) {
   echo 'Фрагмент: ', $arr[2][0] ?? '', '<br>';
   echo 'Смещение: ', $arr[2][1] ?? '';
} // Фрагмент: Текст<br>Смещение: 3
print_r($arr);
// Array (
// [0] => Array ( [0] => <b>Текст</b> [1] => 0 )
// [1] => Array ( [0] => b [1] => 1 )
// [2] => Array ( [0] => Текст [1] => 3 ) )
```

> ***Внимание!***
>
> При использовании флага `PREG_OFFSET_CAPTURE` меняется формат массива `$arr`. Кроме того, смещение указывается в байтах, а не в символах. При использовании кодировки UTF-8 это будет иметь значение.

5.8.4. Поиск всех совпадений с шаблоном

Функция `preg_match()` находит только первое совпадение с шаблоном. Чтобы получить все совпадения, нужно воспользоваться функцией `preg_match_all()`. Формат функции:

```
preg_match_all(string $pattern, string $subject[, array &$matches[,
               int $flags=PREG_PATTERN_ORDER[,
               int $offset=0]]]) : int
```

В первом параметре указывается шаблон регулярного выражения, а во втором — строка, с которой выполняется сравнение. Функция возвращает число найденных совпадений с шаблоном (число `0`, если совпадения не найдены) или значение `false` — в случае ошибки.

Если в параметре `$flags` указано значение `PREG_PATTERN_ORDER` (значение по умолчанию), то массив совпадений `$matches` будет содержать элементы, упорядоченные по порядковому номеру фрагмента, заключенного в шаблоне в круглые скобки. Нулевой элемент массива будет содержать список полных совпадений с шаблоном.

В качестве примера получим все значения между тегами `<b>` и `</b>`:

```
$str = '<b>Текст1</b>Текст2<b>Текст3</b>';
$pattern = '#<b>(.*?)</b>#isu';
$arr = array();
preg_match_all($pattern, $str, $arr, PREG_PATTERN_ORDER);
print_r($arr);
// Array (
// [0] => Array ( [0] => <b>Текст1</b> [1] => <b>Текст3</b> )
// [1] => Array ( [0] => Текст1 [1] => Текст3 ) )
```

Обратите внимание: чтобы ограничить «жадность» квантификатора, мы указали после символа `*` символ `?`.

Если в параметре `$flags` указано значение `PREG_SET_ORDER`, то массив совпадений будет содержать элементы, упорядоченные по номеру совпадения. Каждый элемент массива будет содержать список совпадений фрагментов, заключенных в шаблоне в круглые скобки. Нулевой элемент массива будет содержать полное совпадение с шаблоном:

```
$str = '<b>Текст1</b>Текст2<b>Текст3</b>';
$pattern = '#<b>(.*?)</b>#isu';
$arr = array();
preg_match_all($pattern, $str, $arr, PREG_SET_ORDER);
foreach ($arr as $v) {
   echo $v[1] . " ";
} // Текст1 Текст3
print_r($arr);
// Array (
// [0] => Array ( [0] => <b>Текст1</b> [1] => Текст1 )
// [1] => Array ( [0] => <b>Текст3</b> [1] => Текст3 ) )
```

Если внутри шаблона находятся именованные группы, то массив будет дополнительно содержать элементы с именами групп:

```
$pattern = '#<b>(?<text>.*?)</b>#isu';
```

Результат:

```
Array (
 [0] => Array ( [0] => <b>Текст1</b> [text] => Текст1 [1] => Текст1)
 [1] => Array ( [0] => <b>Текст3</b> [text] => Текст3 [1] => Текст3))
```

Если к флагам PREG_PATTERN_ORDER и PREG_SET_ORDER добавить значение PREG_OFFSET_CAPTURE, то для каждого найденного фрагмента будет указана его позиция в исходной строке:

```
$str = '<b>Текст</b>';
$pattern = '#<b>(.*?)</b>#isu';
$arr = array();
preg_match_all($pattern, $str, $arr,
               PREG_SET_ORDER | PREG_OFFSET_CAPTURE);
print_r($arr);
// Array (
//   [0] => Array (
//          [0] => Array ( [0] => <b>Текст</b> [1] => 0 )
//          [1] => Array ( [0] => Текст [1] => 3 ) ) )
$arr = array();
preg_match_all($pattern, $str, $arr,
               PREG_PATTERN_ORDER | PREG_OFFSET_CAPTURE);
print_r($arr);
// Array (
//    [0] => Array (
//           [0] => Array ( [0] => <b>Текст</b> [1] => 0 ) )
//    [1] => Array (
//           [0] => Array ( [0] => Текст [1] => 3 ) ) )
```

> ***Внимание!***
>
> При использовании флага PREG_OFFSET_CAPTURE меняется формат массива $arr. Кроме того, смещение указывается в байтах, а не в символах. При использовании кодировки UTF-8 это будет иметь значение.

5.8.5. Замена в строке

Функция preg_replace() ищет в строке $subject все совпадения с шаблоном $pattern и заменяет их указанным значением $replacement. Формат функции:

```
preg_replace(mixed $pattern, mixed $replacement, mixed $subject[,
             int $limit=-1[, int &$count]]) : mixed
```

Первые три параметра могут быть одномерными массивами. Если указан параметр $limit, то функция заменит только указанное число первых совпадений с шабло-

ном. Функция возвращает измененную строку или массив. Если совпадения не найдены, то функция вернет исходную строку или массив. При ошибке возвращается значение `null`.

Если указан параметр `$count`, то через него будет доступно количество произведенных замен:

```
$str = '201, 202, 203, 204, 205';
$pattern = '#20[14]#u';
$repl = '111';
$count = 0;
$str2 = preg_replace($pattern, $repl, $str, -1, $count);
echo $str2; // 111, 202, 203, 111, 205
echo "\n Количество замен: $count";
// Количество замен: 2
```

В строке для замены могут присутствовать обратные ссылки `$номер`, соответствующие группам внутри шаблона. В качестве примера возьмем два тега и поменяем имена тегов местами:

```
$str = '<br><td>';
$pattern = '#<([a-z]+)><([a-z]+)>#isu';
$repl = '<$2><$1>';
$str2 = preg_replace($pattern, $repl, $str);
echo htmlspecialchars($str2, ENT_COMPAT | ENT_HTML5, 'UTF-8');
// Выведет в окне Web-браузера: <td><br>
```

Чтобы отделить номер скобки от последующего текста, необходимо заключить номер в фигурные скобки (`${номер}`):

```
$repl = '<${2}><${1}>';
```

Обратиться к найденному фрагменту в круглых скобках можно не только с помощью синтаксиса `$номер` (или `${номер}`), но и указав номер скобок, перед которым стоит слеш (`\номер`):

```
$repl = '<\\2><\\1>';
```

Функция `preg_replace_callback()` выполняет поиск в строке `$subject` по шаблону `$pattern` и замену с использованием функции обратного вызова `$callback`. Формат функции:

```
preg_replace_callback(mixed $pattern, callable $callback,
                      mixed $subject[, int $limit=-1[,
                      int &$count]]) : mixed
```

В отличие от `preg_replace()`, функция `preg_replace_callback()` передает функции, указанной в параметре `$callback`, найденные совпадения в виде массива. Результат, возвращаемый функцией `$callback`, служит фрагментом для замены.

Переделаем наш предыдущий пример и добавим функцию обратного вызова:

```
$str = '<BR><TD>';
$pattern = '#<([a-z]+)><([a-z]+)>#isu';
```

```
$str2 = preg_replace_callback($pattern,
   function ($matches) {
      print_r($matches);
      // Array ( [0] => <BR><TD> [1] => BR [2] => TD )
      $repl = '<' . mb_strtolower($matches[2]) . '><';
      $repl .=  mb_strtolower($matches[1]) . '>';
      return $repl;
   }, $str);
echo htmlspecialchars($str2, ENT_COMPAT | ENT_HTML5,
                      'UTF-8');
// Выведет в окне Web-браузера: <td><br>
```

Нулевой элемент массива `$matches` будет содержать полное соответствие шаблону, а последующие элементы — фрагменты, заключенные в шаблоне в круглые скобки.

5.8.6. Функция *preg_split()*

Функция `preg_split()` разбивает строку `$subject` по шаблону `$pattern` и возвращает массив подстрок. Формат функции:

```
preg_split(string $pattern, string $subject[, int $limit=-1[,
           int $flags=0]]) : array
```

Если задан параметр `$limit`, то функция возвратит только указанное число подстрок. Последняя подстрока будет содержать оставшуюся часть строки:

```
$str = '201, 202, 203, 204, 205';
$pattern = '#[\\s,]+#su';
$arr = preg_split($pattern, $str, 4);
foreach ($arr as $v) {
   echo $v . ';';
} // 201;202;203;204, 205;
```

В параметре `$flags` могут быть указаны следующие значения (или комбинация значений, соединенных оператором `|`):

- `PREG_SPLIT_NO_EMPTY` — функция вернет только непустые подстроки;
- `PREG_SPLIT_DELIM_CAPTURE` — фрагмент, заключенный в шаблоне в круглые скобки, также будет возвращаться;
- `PREG_SPLIT_OFFSET_CAPTURE` — для каждой найденной подстроки будет указана ее позиция в исходной строке.

Пример:

```
$str = '201 - 202, 203. 204';
$pattern = '#([\\s,.-]+)#su';
$arr = preg_split($pattern, $str, -1, PREG_SPLIT_DELIM_CAPTURE);
print_r($arr);
```

Результат:

```
Array
(
    [0] => 201
    [1] =>  -
    [2] => 202
    [3] => ,
    [4] => 203
    [5] => .
    [6] => 204
)
```

Превратить строку в массив символов можно так:

```
$str = 'строка';
$arr = preg_split('##u', $str, -1, PREG_SPLIT_NO_EMPTY);
print_r($arr);
// Array ( [0] => с [1] => т [2] => р [3] => о [4] => к [5] => а )
```

> **Внимание!**
>
> Если не требуется указания шаблона, то вместо функции `preg_split()` лучше использовать функцию `explode()` (см. *разд. 5.7.16*).

5.8.7. Функция *preg_grep()*

Функция `preg_grep()` возвращает новый массив, состоящий из элементов массива `$input`, которые соответствуют шаблону `$pattern`. Формат функции:

```
preg_grep(string $pattern, array $input[, int $flags=0]) : array
```

Индексы массива сохраняются. Если в параметре `$flags` указан флаг `PREG_GREP_INVERT`, то возвращается массив значений, не соответствующих шаблону. В качестве примера получим все элементы массива, состоящие только из цифр, и наоборот:

```
$arr = array(20, 54, 'Текст', 457);
$pattern = '#^[0-9]+$#su';
$arr2 = preg_grep($pattern, $arr);
print_r($arr2);
// Array ( [0] => 20 [1] => 54 [3] => 457 )
$arr3 = preg_grep($pattern, $arr, PREG_GREP_INVERT);
print_r($arr3);
// Array ( [2] => Текст )
```

5.9. Работа с датой и временем

При работе с датой и временем нужно вначале задать зону местного времени. Для этого используется функция `date_default_timezone_set(<Зона>)`:

```
date_default_timezone_set('Europe/Moscow');
```

Задать значение можно также с помощью директивы `date.timezone`:

```
date.timezone=Europe/Moscow
```

Получить текущее значение позволяет функция `date_default_timezone_get()`:

```
echo date_default_timezone_get(); // Europe/Moscow
```

5.9.1. Получение текущих даты и времени

Для получения текущих даты и времени предназначены следующие функции:

- `time()` — возвращает число секунд, прошедшее с 1 января 1970 г.:

  ```
  echo time(); // 1507005425
  ```

- `localtime()` — возвращает локальное время. Формат функции:

  ```
  localtime([int $timestamp=time()[, bool $is_associative=false]]) : array
  ```

  Если первый параметр не указан, то используется значение, возвращаемое функцией `time()`. Если второй параметр имеет значение `true`, то возвращается ассоциативный массив, в противном случае — список:

  ```
  print_r(localtime());
  /*
  Array
  (   [0] => 49
      [1] => 52
      [2] => 0
      [3] => 4
      [4] => 9
      [5] => 117
      [6] => 3
      [7] => 276
      [8] => 0  )
   */
  print_r(localtime(time(), true));
  /*
  Array
  (   [tm_sec] => 49
      [tm_min] => 52
      [tm_hour] => 0
      [tm_mday] => 4
      [tm_mon] => 9
      [tm_year] => 117
      [tm_wday] => 3
      [tm_yday] => 276
      [tm_isdst] => 0  )
  */
  ```

Элементы массива имеют следующие значения:

- `tm_sec` или `0` — секунды (от `0` до `59`);
- `tm_min` или `1` — минуты (от `0` до `59`);
- `tm_hour` или `2` — часы в 24-часовом формате (от `0` до `23`);
- `tm_mday` или `3` — день месяца (от `1` до `31`);
- `tm_mon` или `4` — месяц (`0` — для января и `11` — для декабря);
- `tm_year` или `5` — число лет, прошедшее с 1900 года;
- `tm_wday` или `6` — день недели (`0` — для воскресенья и `6` — для субботы);
- `tm_yday` или `7` — день с начала года (от `0` до `365`);
- `tm_isdst` или `8` — признак летнего времени.

5.9.2. Форматирование даты и времени

Функция `date(<Строка формата>[, <Исходная дата>])` форматирует дату и время в соответствии со строкой формата. Если второй параметр не указан, то используется значение, возвращаемое функцией `time()`. В параметре `<Строка формата>` могут быть использованы следующие служебные символы:

- `U` — число секунд, прошедшее с 1 января 1970 г.;
- `Y` — год из 4 цифр;
- `y` — год из 2 цифр;
- `z` — день с начала года (от `0` до `365`);
- `F` — название месяца по-английски;
- `m` — номер месяца с предваряющим нулем (от `01` до `12`);
- `n` — номер месяца без предваряющего нуля (от `1` до `12`);
- `M` — аббревиатура месяца из трех букв по-английски;
- `t` — количество дней в месяце;
- `d` — номер дня с предваряющим нулем (от `01` до `31`);
- `j` — номер дня без предваряющего нуля (от `1` до `31`);
- `l` — название дня недели по-английски;
- `w` — номер дня недели (`0` — для воскресенья и `6` — для субботы);
- `N` — номер дня недели (`1` — для понедельника и `7` — для воскресенья);
- `D` — аббревиатура дня недели из трех букв по-английски;
- `A` — `AM` (до полудня) или `PM` (после полудня);
- `a` — `am` (до полудня) или `pm` (после полудня);
- `H` — часы в 24-часовом формате (от `00` до `23`);

- `G` — часы в 24-часовом формате без предваряющего нуля (от `0` до `23`);
- `h` — часы в 12-часовом формате (от `01` до `12`);
- `g` — часы в 12-часовом формате без предваряющего нуля (от `1` до `12`);
- `i` — минуты (от `00` до `59`);
- `s` — секунды (от `00` до `59`);
- `P` — смещение временной зоны (в формате `+03:00`);
- `O` — смещение временной зоны (в формате `+0300`);
- `e` — название временной зоны (например, `Europe/Moscow`).

Выведем текущие дату и время таким образом, чтобы день недели и месяц были написаны по-русски (листинг 5.23).

Листинг 5.23. Вывод текущей даты

```
<?php
date_default_timezone_set('Europe/Moscow');
$days = array('воскресенье', 'понедельник', 'вторник', 'среда',
              'четверг', 'пятница', 'суббота');
$months = array('', 'января', 'февраля', 'марта', 'апреля', 'мая',
                'июня', 'июля', 'августа', 'сентября', 'октября',
                'ноября', 'декабря');
$date = 'Сегодня<br>';
$date .= $days[(int)date('w')];
$date .= date(' d ');
$date .= $months[(int)date('n')];
$date .= date(' Y');
$date .= date(' H:i:s') . '<br>' . date('d.m.y');
echo $date;
```

Код, приведенный в листинге 5.23, выведет в окне Web-браузера:

```
Сегодня
пятница 05 января 2018 08:11:37
05.01.18
```

Если в функции `date()` указан второй параметр, то дата будет не текущая, а соответствующая значению из второго параметра. Например, в функцию можно передать число секунд, прошедшее с 1 января 1970 г., и получить любое другое форматирование даты:

```
$date = date("Дата d-m-Y", 1580986651);
echo $date; // Выведет: Дата 06-02-2020
```

Или можно получить дату создания файла:

```
$date = date("Дата d-m-Y", filectime("index.php"));
echo $date; // Выведет: Дата 03-01-2018
```

Функция `strftime()` позволяет отформатировать дату в зависимости от настроек локали. Формат функции:

```
strftime(string $format[, int $timestamp=time()]) : string
```

Если второй параметр не указан, то используется значение, возвращаемое функцией `time()`. В параметре `$format` могут быть указаны следующие служебные символы:

- `%y` — год из двух цифр;
- `%Y` — год из четырех цифр;
- `%j` — день с начала года (от `001` до `366`);
- `%m` — номер месяца с предваряющим нулем (от `01` до `12`);
- `%b` — сокращенное название месяца в текущей локали:

```
setlocale(LC_TIME, 'ru_RU', 'Russian_Russia');
echo iconv('windows-1251', 'UTF-8', strftime("%b")); // окт
```

- `%B` — полное название месяца в текущей локали:

```
setlocale(LC_TIME, 'ru_RU', 'Russian_Russia');
echo iconv('windows-1251', 'UTF-8', strftime("%B"));
// Октябрь
```

- `%W` — номер недели в году. Первый понедельник года считается первым днем первой недели;
- `%d` — номер дня с предваряющим нулем (от `01` до `31`);
- `%w` — номер дня недели (`0` — для воскресенья и `6` — для субботы);
- `%a` — сокращенное название дня недели в текущей локали:

```
setlocale(LC_TIME, 'ru_RU', 'Russian_Russia');
echo iconv('windows-1251', 'UTF-8', strftime("%a")); // Ср
```

- `%A` — полное название дня недели в текущей локали:

```
setlocale(LC_TIME, 'ru_RU', 'Russian_Russia');
echo iconv('windows-1251', 'UTF-8', strftime("%A"));
// среда
```

- `%H` — часы в 24-часовом формате (от `00` до `23`);
- `%I` — часы в 12-часовом формате (от `01` до `12`);
- `%M` — минуты (от `00` до `59`);
- `%S` — секунды (от `00` до `59`);
- `%c` — формат даты и времени по умолчанию в текущей локали:

```
setlocale(LC_TIME, 'ru_RU', 'Russian_Russia');
echo iconv('windows-1251', 'UTF-8', strftime("%c"));
// 07.02.2018 1:42:16
```

- `%x` — представление даты в текущей локали:

```
setlocale(LC_TIME, 'ru_RU', 'Russian_Russia');
echo iconv('windows-1251', 'UTF-8', strftime("%x"));
// 07.02.2018
```

- `%X` — представление времени в текущей локали:

```
setlocale(LC_TIME, 'ru_RU', 'Russian_Russia');
echo iconv('windows-1251', 'UTF-8', strftime("%X"));
// 2:02:30
```

- `%T` — комбинация `%H:%M:%S`:

```
setlocale(LC_TIME, 'ru_RU', 'Russian_Russia');
echo iconv('windows-1251', 'UTF-8', strftime("%T"));
// 02:02:31
```

- `%%` — символ `%`.

Пример вывода текущих даты и времени:

```
setlocale(LC_TIME, 'ru_RU', 'Russian_Russia');
echo iconv('windows-1251', 'UTF-8', strftime("%A %d %B %Y %T"));
// среда 07 Февраль 2018 02:06:27
```

Результаты показаны для следующей локали в Windows:

```
echo setlocale(LC_TIME, 'ru_RU', 'Russian_Russia');
// Russian_Russia.1251
```

Если на вашем компьютере локаль настроена на кодировку UTF-8, то функцию `iconv()` использовать не нужно.

5.9.3. Проверка корректности введенной даты

Функция `checkdate()` позволяет проверить корректность введенной даты. Формат функции:

```
checkdate(<Месяц>, <День>, <Год>)
```

Функция возвращает `true`, если дата, заданная аргументами, является правильной. Дата считается правильной, если:

- год в диапазоне от `1` до `32 767` включительно;
- месяц в диапазоне от `1` до `12` включительно;
- день является допустимым номером дня для месяца, заданного аргументом `<Месяц>`.

Пример проверки:

```
if (checkdate(2, 29, 2018)) echo 'Дата правильная';
else echo 'Нет'; // Т. к. даты 29.02.2018 нет, выведет: Нет
```

5.9.4. Класс *DateTime*

С помощью класса `DateTime` можно получить текущие дату и время, выполнить форматирование, прибавить или вычесть интервал, сравнить два объекта, а также преобразовать строку с датой и временем в объект класса `DateTime`.

Создание объекта

Формат конструктора класса `DateTime`:

```
public __construct([string $time='now'[, DateTimezone $timezone=null]])
```

В первом параметре можно указать дату и время в виде строки. Если параметр не задан, то используется значение `now`, которое означает текущее время. Во втором параметре можно указать зону местного времени.

Пример создания объекта и получения текущих даты и времени:

```
$dt = new DateTime('now', new DateTimezone('Europe/Moscow'));
echo $dt->getTimestamp();              // 1507077740
echo "\n", $dt->format('d.m.Y H:i:s'); // 04.10.2017 03:42:20
```

Пример указания произвольной даты и времени:

```
$dt = new DateTime('2017-10-03 03:15:00',
                   new DateTimezone('Europe/Moscow'));
echo $dt->format('d.m.Y H:i:s');       // 03.10.2017 03:15:00
```

Указание и получение значений

Задать произвольные значения позволяют следующие методы:

- `setTimezone()` — задает зону местного времени. Формат метода:

  ```
  public setTimezone(DateTimezone $timezone) : DateTime
  ```

- `setDate()` — задает дату. Формат метода:

  ```
  public setDate(int $year, int $month, int $day) : DateTime
  ```

- `setTime()` — задает время. Формат метода:

  ```
  public setTime(int $hour, int $minute[, int $second=0[,
                 int $microseconds=0]]) : DateTime
  ```

- `setTimestamp()` — задает число секунд, прошедшее с 1 января 1970 г. Формат метода:

  ```
  public setTimestamp(int $timestamp) : DateTime
  ```

Для получения значений предназначены такие методы:

- `getTimezone()` — возвращает зону местного времени. Формат метода:

  ```
  public getTimezone() : DateTimezone
  ```

- `getOffset()` — возвращает смещение часовой зоны в секундах. Формат метода:

  ```
  public getOffset() : int
  ```

- `getTimestamp()` — возвращает число секунд, прошедшее с 1 января 1970 г. Формат метода:

  ```
  public getTimestamp() : int
  ```

Пример:

```
$dt = new DateTime();
$dt->setTimezone(new DateTimezone('Europe/Moscow'));
$dt->setDate(2017, 10, 3);
$dt->setTime(3, 15, 0);
echo $dt->getTimestamp();                  // 1506989700
echo "\n", $dt->format('d.m.Y H:i:s');     // 03.10.2017 03:15:00
echo "\n", $dt->getTimezone()->getName(); // Europe/Moscow
echo "\n", $dt->getOffset();               // 10800
```

Форматирование строки с датой и временем

В предыдущем примере для вывода даты и времени мы воспользовались методом `format()`, который позволяет выполнить форматирование в соответствии со строкой формата `$format`. Формат метода:

```
public format(string $format) : string
```

В строке `$format` используются служебные символы, применяемые в функции `date()` (см. *разд. 5.9.2*).

Вместо строки формата можно указать следующие константы:

```
echo DateTime::ATOM;            // Y-m-d\TH:i:sP
echo "\n", DateTime::COOKIE;  // l, d-M-Y H:i:s T
echo "\n", DateTime::RSS;     // D, d M Y H:i:s O
echo "\n", DateTime::W3C;     // Y-m-d\TH:i:sP
echo "\n", DateTime::ISO8601; // Y-m-d\TH:i:sO
echo "\n", DateTime::RFC822;  // D, d M y H:i:s O
echo "\n", DateTime::RFC850;  // l, d-M-y H:i:s T
echo "\n", DateTime::RFC1036; // D, d M y H:i:s O
echo "\n", DateTime::RFC1123; // D, d M Y H:i:s O
echo "\n", DateTime::RFC2822; // D, d M Y H:i:s O
echo "\n", DateTime::RFC3339; // Y-m-d\TH:i:sP
```

Пример:

```
$dt = new DateTime();
echo $dt->format(DateTime::COOKIE);
// Wednesday, 07-Feb-2018 04:35:52 MSK
```

Разбор строки с датой и временем

Преобразовать строку с датой и временем в объект класса `DateTime` позволяет статический метод `createFromFormat()`. Формат метода:

```
public static createFromFormat(string $format, string $time[,
                    DateTimezone $timezone]) : DateTime
```

Если преобразовать не удалось, то метод вернет значение `false`. Получить описание ошибки можно с помощью статического метода `getLastErrors()`. Пример:

```
$dt = DateTime::createFromFormat(DateTime::COOKIE,
                'Wednesday, 07-Feb-2018 04:35:52 MSK');
if ($dt !== false) {
   echo $dt->format('d.m.Y H:i:s');    // 07.02.2018 04:35:52
}
else {
   echo "Не удалось преобразовать\n";
   print_r(DateTime::getLastErrors());
}
```

Прибавление и вычитание интервала

Класс `DateTime` содержит методы `add()` и `sub()`, которые позволяют прибавить или вычесть интервал соответственно. Форматы методов:

```
public add(DateInterval $interval) : DateTime
public sub(DateInterval $interval) : DateTime
```

Интервал задается в виде объекта класса `DateInterval`. Формат конструктора:

```
public __construct(string $interval)
```

Конструктор принимает строку специального формата. Первой буквой в строке указывается `P`. Далее приводятся значения компонентов даты. Затем идет буква `T`, за которой приводятся значения компонентов времени. Компоненты даты и времени обозначаются следующими буквами:

- `Y` — год;
- `M` — месяц;
- `D` — день;
- `W` — неделя;
- `H` — час;
- `M` — минута;
- `S` — секунда.

Если не удалось распознать формат строки, то генерируется исключение. Поэтому инструкцию следует разместить внутри блока `try`:

```
$dt = new DateTime('2017-10-03 03:15:00',
                   new DateTimezone('Europe/Moscow'));
echo $dt->format('d.m.Y H:i:s');          // 03.10.2017 03:15:00
try {
   $dt->add(new DateInterval('P1Y2M3DT4H5M6S'));
   // + 1 год 2 месяца 3 дня 4 часа 5 минут и 6 секунд
   echo "\n", $dt->format('d.m.Y H:i:s'); // 06.12.2018 07:20:06
} catch (Exception $e) {
   echo "\n Не удалось прибавить";
}
```

```
try {
   $dt->sub(new DateInterval('P2W'));
   // - 2 недели
   echo "\n", $dt->format('d.m.Y H:i:s'); // 22.11.2018 07:20:06
} catch (Exception $e) {
   echo "\n Не удалось вычесть";
}
```

Вычисление разницы между датами

Получить разницу между двумя датами позволяет метод `diff()`. Формат метода:

```
public diff(DateTimeInterface $object[,
            bool $absolute=false]) : DateInterval
```

Вот пример получения разницы в днях с указанием знака перед отрицательным значением:

```
$dt1 = new DateTime('2017-10-03');
$dt2 = new DateTime('2017-10-23');
$dt3 = new DateTime('2017-09-03');
$interval = $dt1->diff($dt2);
echo $interval->format('Разница %r%a дней');
// Разница 20 дней
$interval = $dt1->diff($dt3);
echo "\n", $interval->format('Разница %r%a дней');
// Разница -30 дней
```

Сравнение двух объектов *DateTime*

Два объекта `DateTime` можно сравнивать с помощью операторов сравнения:

```
$dt1 = new DateTime('2017-10-03');
$dt2 = new DateTime('2017-10-03');
$dt3 = new DateTime('2017-09-03');
var_dump($dt1 == $dt2);  // bool(true)
var_dump($dt1 > $dt3);   // bool(true)
var_dump($dt1 < $dt3);   // bool(false)
```

5.9.5. «Засыпание» программы

Функции `sleep()` и `usleep()` прерывают выполнение сценария на указанное время, по истечении которого сценарий продолжит работу. Форматы методов:

```
sleep(<Время в секундах>)
usleep(<Время в микросекундах>)
```

Пример:

```
echo date('H:i:s');         // 07:12:30
sleep(5);                   // 5 секунд
usleep(5000000);            // 5 секунд
echo "\n", date('H:i:s');   // 07:12:40
```

5.9.6. Измерение времени выполнения

Изменить время выполнения программы позволяет функция `microtime()`, которая возвращает число микросекунд, прошедшее с 1 января 1970 г. Формат функции:

```
microtime([bool $get_as_float=false]) : mixed
```

Если параметр имеет значение `false` или не указан, то функция возвращает строку в формате `<микросекунды> <секунды>`. Параметр `<секунды>` содержит число секунд, прошедшее с 1 января 1970 г., а параметр `<микросекунды>` — число микросекунд, прошедшее после значения параметра `<секунды>`:

```
echo microtime();      // 0.65895300 1507093522
list($micro, $second) = explode(' ', microtime());
$micro = (float)$micro;
$second = (float)$second;
var_dump($micro);     // float(0.658983)
var_dump($second);    // float(1507093522)
```

Если параметр имеет значение `true`, то функция возвращает число типа `float`:

```
echo microtime(true); // 1507093522.6591
```

Создадим имитацию выполнения какого-либо процесса и измерим время его выполнения (листинг 5.24).

Листинг 5.24. Измерение времени выполнения

```
<?php
// Отключаем буферизацию
if (ob_get_level() > 0) ob_end_flush();
ob_implicit_flush();
// Запоминаем начальное время
$t = microtime(true);
echo str_pad("0%<br>\n", 4096);
for ($i = 10; $i < 101; $i += 10) {
   sleep(1);           // Имитация процесса
   echo str_pad("{$i}%<br>\n", 4096);
}
// Вычисляем разницу между метками
printf("%.6f<br>\n", microtime(true) - $t);  // 10.151989
$t2 = $_SERVER['REQUEST_TIME_FLOAT'] ?? 0;
printf("%.6f<br>\n", microtime(true) - $t2); // 10.157440
```

В результате с периодичностью в секунду программа выводит в Web-браузер строки с индикацией хода выполнения, и в самом конце мы получим два числа:

```
10.151989
10.157440
```

Первое число отображает разницу между двумя запросами системного времени, которые мы делали с помощью функции `microtime()`. Если мы вычтем первую метку из второй, то получим время выполнения программы.

Второе число является разницей между текущим значением функции `microtime()` и значением переменной окружения `$_SERVER['REQUEST_TIME_FLOAT']`, которая создается автоматически и содержит метку начала запроса.

С помощью функции `microtime()` мы можем измерять время выполнения фрагментов кода и на основании результата производить различные оптимизации. Например, написали один алгоритм, измерили, затем переписали его иначе, опять измерили. Сравнили два результата и оставили в программе самый эффективный алгоритм. С помощью этой функции можно также искать фрагменты кода, которые выполняются слишком медленно, и пытаться выполнить оптимизацию этих фрагментов, что-либо изменяя в программе.

5.10. Пользовательские функции

Функция — это фрагмент кода PHP, который можно вызвать из любого места программы. В предыдущих разделах мы уже не один раз использовали встроенные функции PHP — например, с помощью функции `time()` получали число секунд, прошедшее с 1 января 1970 года. В этом разделе мы рассмотрим создание пользовательских функций, которые позволят уменьшить избыточность программного кода и повысить его структурированность.

5.10.1. Создание функции

Функция описывается с помощью ключевого слова `function` по следующей схеме:

```
function <Имя функции>([<Параметры через запятую>]) {
   <Тело функции>
   [return[ <Возвращаемое значение>];]
}
```

Имя функции должно быть уникальным и может содержать только буквы, цифры, символ подчеркивания и не может начинаться с цифры. Основное различие между именами функций и переменных заключается в значении регистра символов. Имена переменных зависят от регистра, а названия функций не зависят.

Например, следующие имена функций одинаковы:

```
$str = '\\"Волга\\"';
echo stripslashes($str); // "Волга"
echo StripSlashes($str); // "Волга"
```

После имени функции в круглых скобках можно указать один или несколько параметров через запятую. Параметров может вообще не быть. В этом случае указываются только круглые скобки.

Между фигурными скобками располагаются инструкции PHP, которые будут исполнены после каждого вызова функции.

Функция может возвращать значение при ее вызове. Возвращаемое значение задается с помощью оператора возврата `return`.

Приведем несколько примеров:

- пример функции без параметров:

```
function print_OK() {
   echo "Сообщение при удачно выполненной операции";
}
```

- пример функции с одним параметром:

```
function print_message($msg) {
   echo $msg;
}
```

- пример функции, возвращающей сумму двух переменных:

```
function sum($x, $y) {
   $z = $x + $y;
   return $z;
}
```

В качестве возвращаемого значения в операторе возврата `return` можно указывать не только имя переменной, но и выражение:

```
function sum($x, $y) {
   return ($x + $y);
}
```

В программе функции можно вызвать следующим образом:

```
print_OK();          // Сообщение при удачно выполненной операции
print_message("Сообщение"); // Сообщение
$var = sum(5, 2);    // Переменной $var будет присвоено значение 7
print_message($var); // 7
```

Инструкции, указанные после оператора `return`, никогда не будут выполнены:

```
function sum($x, $y) {
   return ($x + $y);
   echo "Сообщение"; // Эта инструкция никогда не будет выполнена
}
```

Имя переменной, передающей значение функции, может не совпадать с именем переменной внутри функции:

```
function sum($x, $y) {
   return ($x + $y);
}
$var1 = 5;
$var2 = 2;
$var3 = sum($var1, $var2);
```

Некоторые параметры функции могут быть необязательными. Для этого при определении функции такому параметру необходимо присвоить начальное значение. Например, переделаем наш предыдущий пример и сделаем второй параметр необязательным:

```
function sum($x, $y = 2) {
   return ($x + $y);
}
$a = 5;
$var1 = sum($a);    // Переменной $var1 будет присвоено значение 7
$var2 = sum($a, 5); // Переменной $var2 будет присвоено значение 10
```

Таким образом, если второй параметр не задан, его значение будет равно `2`. Обратите внимание: параметры, для которых указаны значения по умолчанию, должны быть расположены после остальных параметров.

5.10.2. Расположение описаний функций

Обычно описания функций принято располагать в начале файла или в отдельном файле, хотя функции могут находиться в любом месте файла, даже после места вызова функции:

```
$a = 5;
$var = sum($a);    // Переменной $var будет присвоено значение 7
function sum($x, $y = 2) {
   return ($x + $y);
}
```

Описание функции можно разместить внутри фигурных скобок, например внутри условия. В этом случае функция будет создана после проверки условия. До этого момента она не существует, поэтому вызов функции до проверки условия вызовет ошибку:

```
// Ошибка функция print_message() еще не определена
// print_message('Сообщение');
$br = true;
if ( $br ) {
   function print_message($msg) {
      echo "{$msg}<br>";
   }
}
else {
   function print_message($msg) {
      echo $msg;
   }
}
print_message('Сообщение'); // Сообщение<br>
```

Конечно же, в данном случае было бы правильнее создать второй параметр и в зависимости от его значения выводить тег `<br>`:

```
function print_message($msg, $br = false) {
   if ( $br ) echo "{$msg}<br>";
   else       echo $msg;
}
print_message('Сообщение', true); // Сообщение<br>
print_message('Сообщение');       // Сообщение
```

Допускается создание функции внутри другой функции. В этом случае вложенная функция будет создана в глобальной области видимости только после вызова первой функции:

```
function create_func() {
   echo 'Функция create_func()';
   function print_message($msg) {
      echo $msg;
   }
}
// Ошибка функция print_message() еще не определена
// print_message('Сообщение');
create_func();                    // Функция create_func()
print_message('Сообщение');       // Сообщение
// Ошибка повторного создания функции print_message()
// create_func();
```

5.10.3. Операторы *require* и *include*. Выносим функции в отдельный файл

Если функции вынесены в отдельный файл, то подключить его позволяют два оператора: `require` и `include`. Операторы имеют следующий формат:

```
require(<Путь или имя файла>)
require <Путь или имя файла>
include(<Путь или имя файла>)
include <Путь или имя файла>
```

Вынесем функцию `sum()` в отдельный файл C:\xampp\htdocs\script.inc (листинг 5.25) и подключим его с помощью оператора `require` (листинг 5.26).

Листинг 5.25. Содержимое файла script.inc

```
<?php
function sum($x, $y) {
   return ($x + $y);
}
?>
```

Листинг 5.26. Использование оператора `require`

```
<?php
require('script.inc');
// require('C:\\xampp\\htdocs\\script.inc');
$x = 5;
$y = 10;
$var = sum($x, $y);
echo $var;  // 15
```

Создать файл script.inc можно, например, с помощью Notepad++. Следует отметить, что подключаемый файл может иметь любое расширение, хотя общепринято давать подключаемым файлам расширение inc (от include).

Попробуем открыть файл script.inc с помощью Web-браузера, набрав в адресной строке: `http://localhost/script.inc`. Отображаем исходный HTML-код и видим содержимое файла script.inc:

```
<?php
function sum($x, $y) {
   return ($x + $y);
}
?>
```

По этой причине необходимо размещать включаемые файлы в каталоге, доступном только для сценария, но недоступном через Интернет. При установке и настройке PHP мы указали местонахождение включаемых файлов в директиве `include_path` файла php.ini:

```
include_path=".;C:/xampp/php/includes;C:/xampp/php/PEAR"
```

Здесь через точку с запятой указано три места для поиска включаемых файлов:

- . (точка) — в том же каталоге, что и исполняемый файл;
- `C:\xampp\php\includes` — в каталоге includes;
- `C:\xampp\php\PEAR` — в каталоге PEAR.

Иными словами, не найдя включаемого файла в том каталоге, где расположен исполняемый файл, интерпретатор выполнит поиск в каталоге includes (C:\xampp\php\includes), а затем в каталоге PEAR (C:\xampp\php\PEAR).

Можно также сохранить включаемый файл с расширением php (в этом случае исходный PHP-код не будет отображаться в окне Web-браузера, но будет выполнен на стороне сервера) или добавить в файл .htaccess запрет доступа к файлам с расширением inc (в этом случае Web-браузер получит ошибку 403):

```
<Files "*.inc">
   Require all denied
</Files>
```

Если включаемый файл содержит исполняемый код, то указывать PHP-дескрипторы нужно обязательно. Иначе PHP-код будет выведен как обычный текст, а при вызове определенных в нем функций отобразится сообщение об ошибке:

```
function sum($x, $y) { return ($x + $y); }
Fatal error: Uncaught Error: Call to undefined function sum() in
C:\xampp\htdocs\index.php:5 Stack trace: #0 {main} thrown in
C:\xampp\htdocs\index.php on line 5
```

Иными словами, во включаемом файле может и не быть кода PHP. Для примера вынесем верхний колонтитул и функцию `sum()` в файл header.inc (листинг 5.27), а нижний колонтитул — в файл footer.inc (листинг 5.28). Затем подключим эти файлы к основному сценарию (листинг 5.29).

Листинг 5.27. Содержимое файла header.inc

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Функции</title>
</head>
<body>
<?php
function sum($x, $y) {
   return ($x + $y);
}
?>
```

Листинг 5.28. Содержимое файла footer.inc

```
</body>
</html>
```

Листинг 5.29. Размещение HTML-кода во включаемом файле

```
<?php
require("header.inc");
$x = 5;
$y = 10;
$var = sum($x, $y);
echo "<p>Результат: {$var}</p>\n";
require("footer.inc");
```

В листинге 5.30 приведен результирующий HTML-код, сформированный после выполнения программы из листинга 5.29.

Листинг 5.30. Результирующий HTML-код

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Функции</title>
</head>
<body>
<p>Результат: 15</p>
</body>
</html>
```

Таким способом можно сформировать шаблоны для множества страниц. Интерпретатор, встретив оператор `require`, сделает содержимое включаемого файла частью страницы. Если файл не может быть загружен, то оператор генерирует фатальную ошибку, и сценарий прекращает работу.

Вместо оператора `require` можно использовать оператор `include`. Если включаемый файл не найден, оператор выведет сообщение об ошибке, но сценарий будет выполняться далее. Если этот файл содержит функции, то каждый вызов функции из него также будет генерировать ошибку.

Оператор `include` возвращает `true`, если файл загружен, и `false` — в случае ошибки. Подавить сообщение об ошибке можно с помощью оператора `@` (листинг 5.31).

Листинг 5.31. Подавление сообщения об ошибке

```
<?php
if ((@include('header.inc')) == true) {
   $x = 5;
   $y = 10;
   $var = sum($x, $y);
   echo "<p>Результат: {$var}</p>\n";
}
require('footer.inc');
```

5.10.4. Операторы *require_once* и *include_once*

Операторы `require_once` и `include_once` работают точно так же, как `require` и `include`. Однако перед включением файла интерпретатор проверяет, включался ли уже этот файл или нет. Если да, то файл не будет подключен. Операторы имеют следующий формат:

```
require_once(<Путь или имя файла>)
require_once <Путь или имя файла>
include_once(<Путь или имя файла>)
include_once <Путь или имя файла>
```

Пример:

```
<?php
require_once('header.inc');
$var = sum(5, 7);
echo "<p>Результат: {$var}</p>\n";
include_once('footer.inc');
```

Применять операторы `require_once` и `include_once` удобно при разработке больших проектов, т. к. в одном из предшествующих сценариев включаемый файл, возможно, уже был подключен. Подключение одного файла дважды может привести к ошибкам.

5.10.5. Проверка существования функции

Вызов несуществующей функции, а также повторное определение существующей функции, приведет к ошибке. Проверить существование встроенной или пользовательской функции позволяет функция `function_exists()`, имеющая такой формат вызова:

```
function_exists(string $function_name) : bool
```

Название проверяемой функции указывается в виде строки без круглых скобок. Если функция существует, то возвращается значение `true`, в противном случае — значение `false` (листинг 5.32).

Листинг 5.32. Пример использования функции `function_exists()`

```
<form action="<?=$_SERVER['SCRIPT_NAME']?>">
<input type="text" name="name_func">
<input type="submit" value="Проверить">
</form>
<?php
if (isset($_GET['name_func'])) {
   if (function_exists($_GET['name_func'])) {
      echo 'Функция ' . $_GET['name_func'] . ' существует';
   }
   else {
      echo 'Функции ' . $_GET['name_func'] . ' нет';
   }
}
```

5.10.6. Вывод всех доступных сценарию функций

Функция `get_defined_functions()` позволяет получить массив с названиями всех определенных функций. Формат функции:

```
get_defined_functions([bool $exclude_disabled=false]) : array
```

Пример:

```
<?php
function print_message($msg) {
   echo $msg;
}
print_r(get_defined_functions());
```

Примерный результат:

```
Array (
    [internal] => Array
        (
            [0] => zend_version
            [1] => func_num_args
            [2] => func_get_arg
            [3] => func_get_args
            [4] => strlen
            ...
        )
    [user] => Array
        (
            [0] => print_message
        )
)
```

Функция `get_loaded_extensions()` возвращает массив всех загруженных модулей, а `get_extension_funcs()` — массив всех функций в модуле, заданном в качестве параметра. Код листинга 5.33 позволяет получить список всех доступных для сценария функций с группировкой по модулям.

Листинг 5.33. Получение списка всех функций с группировкой по модулям

```
<?php
$extensions = get_loaded_extensions();
foreach ($extensions as $module) {
   echo "<b>{$module}</b><br>\n";
   echo "<ul>\n";
   $functions = get_extension_funcs(strtolower($module));
   foreach ($functions as $name) {
      echo "<li>{$name}()</li>\n";
   }
   echo "</ul>\n";
}
```

5.10.7. Объявление типов параметров

При описании функции перед именем параметра допускается указание типа данных. Если тип не задан, то параметр может принимать данные любого типа:

```
function test($var) {
   var_dump($var);
}
test(10);    // int(10)
test(10.5);  // float(10.5)
test('str'); // string(3) "str"
```

Если перед именем параметра добавить тип, то данные, указанные при вызове функции, будут преобразованы к этому типу. Если преобразовать невозможно, генерируется исключение `TypeError`:

```
function test(int $var) {
   var_dump($var);
}
test(10);       // int(10)
test(10.5);     // int(10)
test('10');     // int(10)
// test('str'); // Fatal error: Uncaught TypeError
```

Помимо указания типа параметра, можно задать тип возвращаемого функцией значения. В этом случае после закрывающей круглой скобки добавляется двоеточие и тип данных:

```
function test($var) : int {
   return $var;
}
var_dump(test(10));        // int(10)
var_dump(test(10.5));      // int(10)
var_dump(test('10'));      // int(10)
// var_dump(test('str')); // Fatal error: Uncaught TypeError
```

Как видно из результата, возвращаемые данные преобразуются к указанному типу. Если преобразование невозможно, то генерируется исключение `TypeError`.

Если тип возвращаемого значения не указан, то функция может возвращать данные любого типа:

```
function test($var) {
   return $var;
}
var_dump(test(10));        // int(10)
var_dump(test(10.5));      // float(10.5)
var_dump(test('10'));      // string(2) "10"
var_dump(test('str'));     // string(3) "str"
```

В описании функции можно указать следующие типы данных:

- `bool` — логический тип данных;
- `int` — целые числа;
- `float` — вещественные числа;
- `string` — строка;

- `array` — массивы;
- `iterable` — объекты, поддерживающие итерации (например, массивы);
- `callable` — функции обратного вызова;
- имя класса или интерфейса.

Если функция ничего не возвращает, то после двоеточия можно указать ключевое слово `void`:

```
function print_message($msg) : void { // Функция ничего не возвращает
   echo $msg;
}
print_message('Сообщение');
```

5.10.8. Строгая типизация

Итак, если перед именем параметра добавить тип, то данные, указанные при вызове функции, по умолчанию будут преобразованы к этому типу. Если преобразование невозможно, то генерируется исключение `TypeError`:

```
function test(int $var) : int {
   var_dump($var);
   return $var;
}
test(10);        // int(10)
test(10.5);      // int(10)
test('10');      // int(10)
```

Однако если включен режим *строгой типизации*, то типы данных преобразовываться не будут. Если тип в описании функции не соответствует типу данных при вызове, сразу генерируется исключение `TypeError`. Аналогичная ситуация будет при указании типа возвращаемого функцией значения. Из этого правила есть одно исключение: данные типа `int` можно передать вместо ожидаемого типа `float`, т. к. преобразование выполняется без потерь.

Включить режим строгой типизации позволяет следующая инструкция, расположенная в самом начале файла сразу после открывающего PHP-дескриптора:

```
declare(strict_types=1);
```

> **Примечание**
>
> Не забывайте сохранять файлы в кодировке UTF-8 без BOM, в противном случае метка порядка байтов (сокращенно BOM) приведет к фатальной ошибке.

Строгая типизация включается для отдельного файла и распространяется на вызовы функций, совершенные из этого файла, а не на функции, объявленные в этом файле. Режим включает дополнительную проверку только для типов `bool`, `int`, `float` и `string`. Невозможность преобразования других типов приведет к исключению `TypeError` вне зависимости от того, включен строгий режим или нет.

Пример:

```
<?php
declare(strict_types=1);

function test(int $var) : int {
   var_dump($var);
   return $var;
}
test(10);        // int(10)
// test(10.5);  // Fatal error: Uncaught TypeError
// test('10');  // Fatal error: Uncaught TypeError
```

Часто значение `null` используется для пропуска параметра или для возврата значения при ошибке. Чтобы иметь возможность передавать или возвращать это значение при включенном режиме строгой типизации, нужно перед типом указать знак вопроса:

```
<?php
declare(strict_types=1);

function test(?int $var) : ?int {
   var_dump($var);
   return $var;
}
test(10);        // int(10)
test(null);      // NULL
```

Может возникнуть вопрос: нужно ли использовать режим строгой типизации? Зачем лишние сложности, если типы могут преобразовываться автоматически? Ответ будет однозначным. На этапе написания и отладки программы режим строгой типизации должен быть включен в обязательном порядке, так же как и вывод всех предупреждающих сообщений! Поверьте, это убережет вас от бессонных ночей при поиске ошибок, которые то проявляются, то нет. Учитесь преобразовывать типы данных явным образом, не полагаясь на автоматический режим. Когда программа написана и отлажена, тогда режим строгой типизации, а также вывод предупреждающих сообщений в Web-браузер можно отключить.

5.10.9. Способы передачи параметров

Как вы уже знаете, после названия функции внутри круглых скобок указываются параметры через запятую. Если функция не принимает параметров, то указываются только круглые скобки. Название параметра является локальной переменной. Эта переменная создается при вызове функции, а после выхода из функции она удаляется. Таким образом, локальная переменная видна только внутри функции и ее значение между вызовами не сохраняется.

При вызове функции указывается ее название, после которого внутри круглых скобок передаются значения. Количество параметров в описании функции должно совпадать с количеством параметров при вызове. Переданные значения присваива-

ются переменным, расположенным в той же позиции в описании функции. Так, при вызове функции `sum(10, 20)` (формат `sum($x, $y)`) переменной `$x` будет присвоено значение `10`, а переменной `$y` — значение `20`. Если функция не принимает параметров, то указываются только круглые скобки.

В функцию передается копия значения переменной, поэтому изменение значения внутри функции не затронет значения исходной переменной. Пример передачи параметра по значению приведен в листинге 5.34.

Листинг 5.34. Передача параметра по значению

```
function test(int $x, array $a) : void {
   $x = $x + 20; // Значение вне функции не сохраняется!
   $a[0] = 55;   // Исходный массив не изменяется!
}
$n = 5;
$arr = [ 10 ];
test($n, $arr);
// Значения не изменились!
print_r($n);    // 5
print_r($arr);  // Array ( [0] => 10 )
```

Если внутри функции нужно иметь возможность изменять значение исходной переменной, то в описании функции перед именем параметра следует указать символ ссылки `&` (листинг 5.35).

Листинг 5.35. Передача параметра по ссылке

```
function test(int &$x, array &$a) : void {
   $x = $x + 20; // Изменит значение исходной переменной!
   $a[0] = 55;   // Исходный массив изменится!
}
$n = 5;
$arr = [ 10 ];
test($n, $arr);
// Значения изменились!
print_r($n);    // 25
print_r($arr);  // Array ( [0] => 55 )
```

5.10.10. Способы возврата значений

При возврате значения из функции ситуация аналогичная — по умолчанию возвращается копия значения (листинг 5.36).

Листинг 5.36. Возврат результата по значению

```
function test(int &$x) : int {
   $x = $x + 20; // Изменит значение исходной переменной!
```

```
   return $x;    // Возвращается копия значения!
}
$n = 5;
$k = test($n);
print_r($n);    // 25
print_r($k);    // 25
$n = 11;
$k = 88;
print_r($n);    // 11
print_r($k);    // 88
```

Чтобы вернуть значение по ссылке, следует перед названием функции указать символ ссылки `&` (листинг 5.37). Но этого недостаточно! Нужно дополнительно указать символ ссылки `&` перед названием функции при вызове.

Листинг 5.37. Возврат результата по ссылке

```
function &test(int &$x) : int {
   $x = $x + 20; // Изменит значение исходной переменной!
   return $x;    // Возвращается ссылка!
}
$n = 5;
$k = &test($n); // Символ & обязателен!
print_r($n);    // 25
print_r($k);    // 25
$n = 11;
$k = 88;
print_r($n);    // 88 (значение изменилось!)
print_r($k);    // 88
```

> ***Примечание***
>
> Используйте способ возврата значения по ссылке только тогда, когда вы понимаете, что делаете. В большинстве случаев лучше применять способ возврата, используемый по умолчанию.

5.10.11. Переменное число параметров в функции

Функции `func_get_args()` и `func_get_arg()` позволяют получить доступ ко всем параметрам, переданным функции, а функция `func_num_args()` дает возможность определить общее число переданных параметров (листинг 5.38).

Листинг 5.38. Определение числа переданных параметров

```
function sum($x, $y) {
   echo 'Число параметров: ' . func_num_args() . "\n";
   echo 'Значения: ';
```

```
   print_r(func_get_args());
   return func_get_arg(0) + func_get_arg(1);
}
echo sum(10, 20); // 30
```

Результат:

```
Число параметров: 2
Значения: Array
(
    [0] => 10
    [1] => 20
)
30
```

Какой в этом смысл? Дело в том, что PHP поддерживает переменное число параметров в функции. При использовании таких функций можно передать нашей функции больше параметров, чем первоначально объявлено. Можно, например, просуммировать сразу несколько чисел, а не только два (листинг 5.39).

Листинг 5.39. Переменное число параметров в функции

```
function sum($x, $y) {
   $result = 0;
   $count = func_num_args();
   for ($i = 0; $i < $count; $i++) {
      $result += func_get_arg($i);
   }
   return $result;
}
echo sum(5, 6, 7, 20); // 38
```

Такой же результат можно получить, используя функцию `func_get_args()` (листинг 5.40).

Листинг 5.40. Использование функции `func_get_args()`

```
function sum($x, $y) {
   $result = 0;
   foreach (func_get_args() as $value) {
      $result += $value;
   }
   return $result;
}
echo sum(5, 6, 7, 20); // 38
```

До версии PHP 5.6 никакого способа обозначить переменное число параметров не было. Но, начиная с версии 5.6, перед названием переменной можно указать много-

точие (...). В этом случае переменная становится массивом, через который доступны все значения (листинг 5.41). Использовать функции `func_get_args()`, `func_get_arg()` и `func_num_args()` в этом случает не нужно. Следует учитывать, что параметр с многоточием должен быть расположен в списке параметров последним.

Листинг 5.41. Переменное число параметров в функции

```
function sum(int $n, int ...$numbers) : int {
   $result = $n;
   foreach ($numbers as $value) {
      $result += $value;
   }
   return $result;
}
echo sum(5);           // 5
echo sum(5, 6, 7, 20); // 38
```

> ***Внимание!***
> Если перед названием параметра указано многоточие, то при вызове значение можно не указывать.

5.10.12. Распаковка массива

Символ многоточия можно также использовать для распаковки массива при передаче значений в функцию при вызове (листинг 5.42).

Листинг 5.42. Распаковка массива при передаче в функцию

```
function sum($x, $y) {
   return $x + $y;
}
$arr = [5, 7];
echo sum(...$arr);       // 12
echo sum(...[3, 8]);     // 11
echo sum(...[3, 8, 9]); // 11
```

Если функция возвращает массив, то для его распаковки можно использовать оператор `list()` (листинг 5.43).

Листинг 5.43. Распаковка при возврате массива

```
function test() : array {
   return array(1, 2, 3);
}
list($x, $y, $z) = test();
```

```
echo $x;    // 1
echo $y;    // 2
echo $z;    // 3
```

5.10.13. Глобальные и локальные переменные

Глобальными называют переменные, объявленные вне функции. В PHP глобальные переменные видны в любой части программы, кроме функций.

Локальные переменные объявляются внутри функции и видны только в теле функции. Переменные, указанные в качестве параметров, также являются локальными. Если имена локальной и глобальной переменной совпадают, то все операции внутри функции осуществляются с локальной переменной, а значение глобальной не изменяется.

Рассмотрим область видимости переменных на примере (листинг 5.44).

Листинг 5.44. Глобальные и локальные переменные

```
function test($z) { // $z локальная переменная
   $x = 5; // Локальная переменная
   echo 'Локальная переменная $x = ' . $x . "<br>\n";
   echo 'Локальная переменная $z = ' . $z . "<br>\n";
   echo 'Глобальная переменная $y = ' . ($y ?? 'NULL');
   echo ", т. е. не видна внутри функции<br>\n";
}
$x = 10;  // Глобальная переменная
echo 'Глобальная переменная $x = ' . $x . "<br>\n";
$y = 7;   // Глобальная переменная
test(33);
echo "Вызов функции<br>\n";
echo 'Глобальная переменная $x осталась = ' . $x . "<br>\n";
echo 'Локальная переменная $z = ' . ($z ?? 'NULL');
echo ', т. е. не видна вне тела функции';
```

В окне Web-браузера получим следующий результат:

```
Глобальная переменная $x = 10
Локальная переменная $x = 5
Локальная переменная $z = 33
Глобальная переменная $y = NULL, т. е. не видна внутри функции
Вызов функции
Глобальная переменная $x осталась = 10
Локальная переменная $z = NULL, т. е. не видна вне тела функции
```

Как видно из результата, переменная `$z`, объявленная в параметре функции `test()`, не доступна вне функции. Объявление внутри функции локальной переменной `$x` не изменило значения одноименной глобальной переменной. А глобальная переменная `$y` не видна внутри функции `test()`.

Для того чтобы глобальная переменная была видна внутри функции, необходимо перед именем переменной в теле функции указать ключевое слово `global`. Продемонстрируем это на примере (листинг 5.45).

Листинг 5.45. Использование глобальных переменных внутри функции

```
function test($z) {
   global $x;
   echo "Глобальная переменная \$x внутри функции = {$x}<br>\n";
   $x += $z;
}
$x = 10;
echo "Глобальная переменная \$x вне функции = {$x}<br>\n";
test(33);
echo "Вызов функции<br>\n";
echo "Глобальная переменная \$x после функции = {$x}<br>\n";
```

В окне Web-браузера получим следующий результат:

```
Глобальная переменная $x вне функции = 10
Глобальная переменная $x внутри функции = 10
Вызов функции
Глобальная переменная $x после функции = 43
```

Внутри функции к глобальной переменной можно обратиться через суперглобальный массив `$GLOBALS` (листинг 5.46).

Листинг 5.46. Использование суперглобального массива `$GLOBALS`

```
function test($z) {
   echo "Глобальная переменная \$x внутри функции = ";
   echo $GLOBALS['x'] . "<br>\n";
   $GLOBALS['x'] += $z;
}
$x = 10;
echo "Глобальная переменная \$x вне функции = {$x}<br>\n";
test(33);
echo "Вызов функции<br>\n";
echo "Глобальная переменная \$x после функции = {$x}<br>\n";
```

В итоге, в окне Web-браузера получим такой же результат:

```
Глобальная переменная $x вне функции = 10
Глобальная переменная $x внутри функции = 10
Вызов функции
Глобальная переменная $x после функции = 43
```

5.10.14. Статические переменные

Если внутри функции объявлена *статическая переменная*, то после завершения работы функции она не будет удалена и сохранит свое значение. Объявляется статическая переменная с помощью ключевого слова `static`, которое указывается перед именем переменной.

Выведем все четные числа от `2` до `100` (листинг 5.47).

Листинг 5.47. Использование статических переменных

```
function test($n) {
   static $var = 0;
   $var += $n;
   echo "{$var}<br>\n";
}
for ($i = 0; $i < 50; $i++) test(2);
```

5.10.15. Анонимные функции

Помимо обычных функций язык PHP позволяет использовать *анонимные функции*. Анонимная функция описывается по следующей схеме:

```
<Переменная> = function ([<Параметры через запятую>])
                        [use (<Переменные>)] : [<Тип>] {
   <Тело функции>
   [return[ <Возвращаемое значение>];]
};
```

Это почти как при описании обычной функции, но анонимная функция не имеет названия, и создаваемый объект присваивается какой-либо переменной, через которую можно вызвать функцию. Обратите внимание: после закрывающей фигурной скобки обязательно нужно указать точку с запятой.

Пример создания и вызова анонимной функции показан в листинге 5.48.

Листинг 5.48. Анонимные функции

```
$sum = function (int $x, int $y) : int {
   return $x + $y;
};
echo $sum(5, 7); // 12
```

Анонимная функция становится видимой только после описания. Внутри блока по умолчанию видны лишь локальные переменные, указатель `$this` и суперглобальные массивы. Для обращения к глобальным переменным нужно воспользоваться ключевым словом `global`. Анонимные функции позволяют захватить значения переменных в родительской области видимости — для этого нужно указать пере-

менные через запятую внутри круглых скобок после ключевого слова `use`. Давайте рассмотрим области видимости анонимных функций на примере (листинг 5.49).

Листинг 5.49. Области видимости при использовании анонимных функций

```
$a = 20; // Значение до описания функции
$b = 33;

$test = function () use($a) : void {
   global $b;
   echo 'Глобальная переменная $a = ' . ($a ?? 'NULL');
   echo ", т. е. значение до описания функции<br>\n";
   echo 'Глобальная переменная $b = ' . ($b ?? 'NULL');
   echo ", т. е. значение после описания функции<br>\n";
};

$a = 2; // Изменили значение после описания функции
$b = 3;
$test();
```

В окне Web-браузера получим следующий результат:

```
Глобальная переменная $a = 20, т. е. значение до описания функции
Глобальная переменная $b = 3, т. е. значение после описания функции
```

Как видно из примера, место описания анонимной функции и место ее вызова могут быть расположены в разных областях видимости. При обращении к глобальной переменной мы видим ее значение на момент вызова, тогда как обращение к переменной в родительской области видимости содержит значение на момент описания функции. Если при вызове нужно иметь доступ к текущему значению переменной, то перед названием переменной нужно указать символ ссылки `&`:

```
$test = function () use(&$a) : void {
   // ...
};
```

Анонимную функцию можно обернуть в круглые скобки и сразу вызвать:

```
$value = (function ($x, $y) {
   return $x + $y;
}) (10, 5);
echo $value; // 15
```

Анонимные функции в языке PHP являются объектами класса `Closure`. Это не сложно заметить, если вывести информацию с помощью функции `var_dump()`:

```
var_dump($test);
```

Результат:

```
object(Closure)#1 (1) {
  ["static"]=>
```

```
  array(1) {
    ["a"]=>
    &int(2)
  }
}
```

5.10.16. Функции обратного вызова

Функцию можно передать другой функции в качестве параметра. Такие функции называются *функциями обратного вызова*.

В качестве типа данных перед названием параметра можно указать ключевое слово `callable`. При вызове название функции задается в виде строки. Вместо строки можно передать анонимную функцию.

Вызов функции осуществляется с помощью круглых скобок, внутри которых передаются значения параметрам, или следующих функций:

```
call_user_func(callable $callback[, mixed $parameter[,
               mixed $...]]) : mixed
call_user_func_array(callable $callback, array $param_arr) : mixed
```

Пример использования функций обратного вызова:

```
function change($x, $y, callable $callback) {
   echo call_user_func($callback, $x, $y);
   echo call_user_func_array($callback, array($x, $y));
}
```

Проверить, может ли переданное значение быть вызвано в качестве функции, позволяет функция `is_callable()`. Функция возвращает значение `true`, если может быть вызвана, и `false` — в противном случае. Формат функции:

```
is_callable(mixed $var[, bool $syntax_only=false[,
            string &$callable_name]]) : bool
```

Еще один пример использования функций обратного вызова показан в листинге 5.50.

Листинг 5.50. Функции обратного вызова

```
function change($x, $y, callable $callback) {
   return $callback($x, $y);
}
$add = function ($a, $b) {
   return $a + $b;
};
function sub($a, $b) {
   return $a - $b;
}
// Передача названия в виде строки
echo change(2, 1, 'sub'), "\n"; // 1
```

```
// Передача анонимной функции
echo change(2, 1, $add), "\n";  // 3
echo change(2, 1, function ($a, $b) {
   return $a * $b;
}); // 2
```

5.10.17. Функции-генераторы

Функцией-генератором называется функция, которая может возвращать одно значение из нескольких значений на каждой итерации. Приостановить выполнение функции и превратить функцию в генератор позволяет ключевое слово `yield`. В качестве примера напишем функцию, которая возводит элементы последовательности в указанную степень (листинг 5.51).

Листинг 5.51. Функции-генераторы

```
function test($x, $y) {
   for ($i = 1; $i <= $x; $i++) {
      yield $i ** $y;
   }
}
$obj = test(10, 2);
foreach ($obj as $value) {
   echo $value, " ";
} // 1 4 9 16 25 36 49 64 81 100
echo "\n";
$obj = test(10, 3);
foreach ($obj as $value) {
   echo $value, " ";
} // 1 8 27 64 125 216 343 512 729 1000
```

После вызова функции-генератора возвращается объект `Generator`, поддерживающий однонаправленные итерации. На каждом шаге возвращается значение, после чего состояние генератора сохраняется. При следующем вызове функции выполнение возобновляется с места, на котором прервалось. Таким способом можно сгенерировать бесконечное число значений (по одному на каждой итерации), не опасаясь превысить лимит оперативной памяти.

Возвращаемый объект содержит метод `current()`, с помощью которого можно получить текущее значение, метод `key()` — возвращающий текущий ключ, метод `next()` — для перехода к следующему значению, а также метод `valid()` — позволяющий проверить наличие значения.

Вот пример использования этих методов:

```
$obj = test(3, 2);
while ($obj->valid()) {
   echo $obj->key() . '=&gt;' . $obj->current(), " ";
```

```
   $obj->next();
} // 0=>1 1=>4 2=>9
```

После ключевого слова `yield` можно указать не только значение, но и пару ключ/значение (листинг 5.52).

Листинг 5.52. Возврат пары ключ/значение

```
function test($x, $y) {
   for ($i = 1; $i <= $x; $i++) {
      yield $i => $i ** $y;
   }
}
$obj = test(5, 2);
foreach ($obj as $key => $value) {
   echo $key . '=&gt;' . $value, " ";
} // 1=>1 2=>4 3=>9 4=>16 5=>25
```

Кроме того, после ключевого слова `yield` может быть указано ключевое слово `from`, после которого задается другая функция-генератор или объект, поддерживающий итерации — например, массив (листинг 5.53).

Листинг 5.53. Инструкция `yield from`

```
function test() {
   yield 1;
   yield from [2, 3, 4];
   yield from test2();
   yield 7;
}
function test2() {
   yield 5;
   yield 6;
}

$obj = test();
foreach ($obj as $value) {
   echo $value, " ";
} // 1 2 3 4 5 6 7
```

Функция `iterator_to_array()` позволяет скопировать значения из объекта, поддерживающего итерации, в массив. Формат функции:

```
iterator_to_array(Traversable $iterator[, bool $use_keys=true]) : array
```

При использовании инструкции `yield from` следует помнить, что она не сбрасывает ключи. В результате некоторые ключи могут пересекаться, что повлечет перезапись

уже записанных значений. Чтобы этого избежать, следует в параметре `$use_keys` явным образом указать значение `false` (листинг 5.54).

Листинг 5.54. Функция `iterator_to_array()`

```
function test() {
   yield 1;
   yield from test2();
   yield 4;
}
function test2() {
   yield 2;
   yield 3;
}
$arr = iterator_to_array(test());
print_r($arr); // Array ( [0] => 2 [1] => 4 )
$arr = iterator_to_array(test(), false);
print_r($arr); // Array ( [0] => 1 [1] => 2 [2] => 3 [3] => 4 )
```

Класс `Generator` содержит метод `send()`, с помощью которого можно передать значение в генератор. Формат метода:

```
public send(mixed $value) : mixed
```

Получить переданное значение внутри функции можно так:

```
$status = yield $i => $i ** $y;
```

В предыдущих версиях PHP нужно было добавлять круглые скобки:

```
$status = (yield $i => $i ** $y);
```

Пример передачи значения в генератор приведен в листинге 5.55.

Листинг 5.55. Передача значения в генератор

```
function test($x, $y) {
   for ($i = 1; $i <= $x; $i++) {
      $status = yield $i => $i ** $y;
      if ($status === true) break;
   }
}
$obj = test(5, 2);
foreach ($obj as $key => $value) {
   echo $key . '=&gt;' . $value, " ";
   if ($key > 2) $obj->send(true);
} // 1=>1 2=>4 3=>9
```

Функция-генератор может возвращать какое-либо значение, которое доступно через метод `getReturn()` (листинг 5.56). Обратите внимание: если поток управления

не дошел до инструкции `return`, то при попытке вызвать метод будет сгенерировано исключение.

Листинг 5.56. Метод `getReturn()`

```
function test($x, $y) {
   for ($i = 1; $i <= $x; $i++) {
      yield $i ** $y;
   }
   return 2;
}
$obj = test(5, 2);
foreach ($obj as $value) {
   echo $value, " ";
} // 1 4 9 16 25
echo $obj->getReturn(); // 2
```

5.10.18. Рекурсия

Рекурсия — это возможность функции вызывать саму себя. При каждом вызове функции создается новый набор локальных переменных. Рекурсию удобно использовать для перебора объекта, имеющего заранее неизвестную структуру, или выполнения неопределенного количества операций. Типичным применением рекурсии является вычисление факториала числа (листинг 5.57).

Листинг 5.57. Вычисление факториала

```
<p><b>Вычисление факториала</b><br><br>
Введите число</p>
<form action="<?=$_SERVER['SCRIPT_NAME']?>">
<input type="text" name="fact">
<input type="submit" value="OK">
</form>
<?php
function factorial(float $x) : float {
   if ($x <= 1) return 1;
   else return ($x * factorial($x - 1));
}
if (isset($_GET['fact'])) {
   $fact = $_GET['fact'];
   if (! preg_match('/^[0-9]+$/su', $fact)) {
      echo 'Необходимо ввести целое число!';
   }
   else {
      echo "Факториал числа {$fact} = " . factorial( (int)$fact );
   }
}
?>
```

5.11. Пространства имен

Предположим, у нас есть два модуля с названиями: module1.inc и module2.inc. Оба модуля содержат следующий код:

```
<?php
function sum($x, $y) {
   return $x + $y;
}
?>
```

Если мы в основной программе попробуем подключить сразу два модуля:

```
<?php
require_once 'module1.inc';
require_once 'module2.inc';
$var = sum(5, 12);
echo "<p>Результат: {$var}</p>\n";
```

то получим сообщение об ошибке: **Fatal error: Cannot redeclare sum()**, т. к. функция `sum()` не может быть объявлена дважды. Подобная ситуация часто случается, когда мы пытаемся использовать чужой код. Что же делать в этом случае — если нужно использовать одноименные функции из разных модулей?

Для решения этой проблемы в PHP версии 5.3 были введены пространства имен.

5.11.1. Объявление пространства имен

Пространство имен объявляется с помощью ключевого слова `namespace` по следующим схемам:

```
namespace <Имя>;
namespace <Имя1>\<Имя2>\<ИмяN>;
namespace <Имя> {
}
namespace <Имя1>\<Имя2>\<ИмяN> {
}
```

> ***Внимание!***
>
> Переменные, объявленные внутри пространства имен, добавляются в глобальное пространство имен, а не в текущее. Пространства имен работают только с классами, интерфейсами, функциями и константами.

Следует учитывать, что инструкция `namespace` должна быть расположена в самом начале файла сразу после открывающего PHP-дескриптора. Исключением является инструкция `declare`, которая должна быть расположена раньше. Обычного текста перед открывающим PHP-дескриптором быть не должно. Не забывайте также сохранять файлы в кодировке UTF-8 без BOM, в противном случае метка порядка байтов (сокращенно BOM) приведет к фатальной ошибке.

Пример объявления пространства имен для модуля module1.inc приведен в листинге 5.58.

Листинг 5.58. Объявление пространства имен (файл module1.inc)

```
<?php
declare(strict_types=1);
namespace module1;

function sum($x, $y) {
   return $x + $y;
}
?>
```

Названия пространств имен могут состоять из нескольких частей. В этом случае названия подпространств приводятся через символ \. Чтобы название пространства имен было уникальным, лучше в названиях подпространств использовать части названия сайта в обратном порядке. Причем в файловой системе лучше создать соответствующую структуру каталогов. Например, файл module2.inc (листинг 5.59) следует разместить в каталоге C:\xampp\php\includes\com\example.

Листинг 5.59. Объявление пространства имен (файл module2.inc)

```
<?php
namespace com\example\module2;

function sum($x, $y) {
   return $x + $y;
}
?>
```

В одном файле допускается объявление сразу нескольких пространств имен. Кроме того, одно пространство имен может быть объявлено сразу в нескольких файлах. В качестве примера создадим модуль module3.inc с кодом из листинга 5.60.

Листинг 5.60. Объявление нескольких пространств имен (файл module3.inc)

```
<?php
namespace module3 {
   function test() {
      echo 'Пространство имен ' . __NAMESPACE__ . "<br>\n";
   }
}
namespace module1 {
   const MY_CONST = 10;
}
```

```
namespace { // Глобальное пространство имен
   function print_message($msg) {
      echo $msg . "<br>\n";
   }
}
?>
```

Обратите внимание: последняя инструкция `namespace` не имеет имени. В этом случае все идентификаторы, объявленные внутри фигурных скобок, добавляются в глобальное пространство имен.

В листинге 5.60 мы также использовали константу `__NAMESPACE__`, которая содержит название текущего пространства имен в виде строки. В глобальном пространстве имен константа содержит пустую строку.

5.11.2. Использование пространств имен

Идентификаторы, добавленные в именованное пространство имен, больше не видны в глобальном пространстве имен внутри основной программы. Единственный идентификатор, к которому мы можем обратиться после подключения всех ранее созданных модулей, — это функция `print_message()`. В остальных случаях необходимо явным образом указать название пространства имен (листинг 5.61).

Листинг 5.61. Использование пространств имен (файл index.php)

```
<?php
require_once 'module1.inc';
require_once 'com\\example\\module2.inc';
require_once 'module3.inc';

print_message('Глобальное пространство');
print_message(\module1\sum(5, 12));           // 17
print_message(\module1\MY_CONST);             // 10
print_message(\com\example\module2\sum(2, 3)); // 5
\module3\test(); // Пространство имен module3
```

Чтобы явно указать, что функция `print_message()` расположена в глобальном пространстве имен, можно перед ее названием добавить символ `\`:

```
\print_message('Глобальное пространство');
```

Это становится особенно полезно, если мы переопределяем встроенную функцию, но хотим вызвать ее внутри нашей функции. Давайте переопределим функцию `date()` внутри пространства имен `module1`:

```
function date() {
   return \date('d.m.y H:i:s');
}
```

Если мы в этом случае не укажем символ \, то получим рекурсивный вызов функции date() из пространства имен module1, что приведет к фатальной ошибке.

Вот пример вызова из основной программы:

```
print_message(\module1\date()); // 08.10.17 09:10:14
```

Если символ \ в начале не указан, то к имени идентификатора автоматически добавляется название текущего пространства имен, поэтому внутри модулей при обращении к идентификаторам можно не указывать название текущего пространства имен. Если в текущем пространстве имен идентификатор отсутствует, то выполняется поиск в глобальном пространстве имен. Исключением являются имена классов, к которым всегда добавляется название текущего пространства имен и поиск в глобальном пространстве не производится. Поэтому перед встроенными классами необходимо указывать символ \:

```
function date() {
   $dt = new \DateTime();
   return $dt->format('d.m.Y H:i:s');
}
```

Добавим в пространство имен module1 функцию test() и внутри нее вызовем функцию sum() из того же пространства имен:

```
function test() {
   return sum(2, 3);
}
```

Вот пример явного указания пространства имен:

```
function test() {
   return \module1\sum(2, 3);
}
```

Для явного указания текущего пространства имен можно также воспользоваться ключевым словом namespace:

```
function test() {
   return namespace\sum(2, 3);
}
```

5.11.3. Инструкция *use*

Что делать, если название пространства имен получилось очень длинным? Неужели такое длинное название нужно указывать перед каждым идентификатором? Для решения этой проблемы существует инструкция use, которая позволяет создать псевдоним или импортировать идентификатор в текущее пространство имен. Формат инструкции use для пространств имен и классов:

```
use <Пространство имен>[\<Класс>][ as <Псевдоним>];
```

Пример создания псевдонима для пространства имен com\example\module2:

```
use com\example\module2 as m2;
```

После этой инструкции мы можем обратиться к функции `sum()` так:

```
print_message(m2\sum(2, 6));
```

Если оператор `as` не указан, то создается короткое имя с названием класса или названием последнего подпространства:

```
use com\example\module2;
```

В этом случае можно использовать имя `module2`:

```
print_message(module2\sum(2, 7));
```

Пример импорта класса из глобального пространства имен:

```
use DateTime;
```

Инструкция `use` позволяет создать псевдонимы сразу для нескольких пространств имен или классов. В этом случае они приводятся через запятую:

```
use module1 as m1, module3 as m3;
```

С помощью инструкции `use` можно импортировать в текущее пространство имен функцию или константу. В этом случае используются следующие форматы:

```
use function <Пространство имен>\<Функция>[ as <Псевдоним>];
use const <Пространство имен>\<Константа>[ as <Псевдоним>];
```

Пример:

```
use function module1\sum;
use const module1\MY_CONST;
```

Если нужно импортировать несколько классов, функций или констант, то удобно воспользоваться следующими форматами инструкции `use`:

```
use <Пространство имен>\{<Класс 1>[ as <Псевдоним 1>],
                         <Класс N>[ as <Псевдоним N>]};
use function <Пространство имен>\{<Функция 1>[ as <Псевдоним 1>],
                                  <Функция N>[ as <Псевдоним N>]};
use const <Пространство имен>\{<Константа 1>[ as <Псевдоним 1>],
                               <Константа N>[ as <Псевдоним N>]};
```

Пример импорта двух функций:

```
use function module1\{date as dt, test};
```

В дальнейшем указывать пространство имен перед названиями функций не надо:

```
print_message(dt());
print_message(test());
```

Полный пример использования инструкции `use` приведен в листинге 5.62.

Листинг 5.62. Инструкция `use` (файл index.php)

```
<?php
namespace main;
```

```
require_once 'module1.inc';
require_once 'com\\example\\module2.inc';
require_once 'module3.inc';
// Импорт классов и создание псевдонимов для пространств имен
use com\example\module2 as m2;
use com\example\module2;
use DateTime;
use module1 as m1, module3 as m3;
// Импорт функций и констант
use function module1\sum;
use const module1\MY_CONST;
use function module1\{date as dt, test};

print_message(sum(5, 12));            // 17
print_message(m1\sum(3, 2));          // 5
print_message(MY_CONST);              // 10
print_message(m2\sum(2, 6));          // 8
print_message(module2\sum(2, 7));     // 9
m3\test(); // Пространство имен module3

$d = new DateTime();
print_message($d->getTimestamp());
print_message(dt());      // 08.10.17 09:10:14
print_message(test());    // 5
```

5.12. Классы и объекты

Объектно-ориентированное программирование (*ООП*) — это способ организации программы, позволяющий использовать один и тот же код многократно. Основным «кирпичиком» ООП является *класс*. Класс включает набор переменных (называемых *полями* или *свойствами* класса) и функций для управления этими переменными (называемых *методами*). В совокупности свойства и методы называются *членами* класса. После создания класса его название становится новым типом данных. Тем самым пользовательские классы расширяют возможности языка PHP.

Основными понятиями ООП являются:

- *инкапсуляция* — сокрытие данных внутри класса;
- *наследование* — возможность создания производных классов на основе базового класса. При этом производный класс автоматически получает возможности базового класса и может добавить новую функциональность или переопределить некоторые методы;
- *полиморфизм* — смысл действия, которое выполняет одноименный метод, зависит от объекта, над которым это действие выполняется.

Что же должно быть представлено в виде классов, а что в виде методов или свойств? Если слово является именем существительным (автомобиль, телевизор,

кнопка и т. д.), то оно может быть описано как *класс*. *Метод* описывает действие, применяемое к объекту — например, начать движение, нажать кнопку, изменить цвет и т. д. *Свойство* предназначено для сохранения текущего состояния объекта и его характеристик — например, размер кнопки и ее цвет, признак, нажата она или нет.

5.12.1. Создание класса и экземпляра класса

Описание класса начинается с ключевого слова `class`:

```
class <Имя класса> {
   // <Свойства и методы класса>
}
```

Пример создания класса с названием `MyClass` в пространстве имен `main` приведен в листинге 5.63.

Листинг 5.63. Создание класса

```
<?php
namespace main;

class MyClass {
   public $x = 10;              // Свойство

   public function toString() { // Метод
      return 'MyClass $x = ' . $this->x;
   }
}
```

На основе класса можно создать множество *объектов* (*экземпляров* класса). При этом для каждого объекта создается свой набор локальных переменных. Класс можно сравнить с чертежом, а объект — с изделием, произведенным по этому чертежу. Чертеж всегда один, а вот изделий может быть бесконечное множество. Причем каждое изделие уникально и не зависит от других изделий.

Для создания экземпляра класса предназначен оператор `new`. После оператора указывается имя класса, к которому будет относиться этот экземпляр, и круглые скобки. Внутри круглых скобок можно передавать некоторые параметры, задавая таким образом начальные значения свойствам класса:

```
<Экземпляр класса> = new <Имя класса>([<Параметры>]);
```

Создадим экземпляр класса `MyClass`:

```
$obj = new MyClass();
```

Если класс расположен в пространстве имен, то перед именем класса можно указать название пространства имен:

```
$obj = new \main\MyClass();
```

Или так:

```
$obj = new namespace\MyClass();
```

Не следует забывать, что глобальные классы не видны внутри пользовательского пространства имен. Нужно обязательно указать символ \ перед именем глобального класса:

```
$dt = new \DateTime();
```

Можно также импортировать имя глобального класса в текущее пространство имен с помощью инструкции `use`:

```
use \DateTime;
$dt = new DateTime();
```

Создать экземпляр класса можно через переменную, содержащую название класса в виде строки. Причем, если класс расположен внутри пространства имен, то перед именем класса обязательно нужно указать название пространства имен:

```
$name = 'main\\MyClass';
$obj = new $name();
```

Получить полное название класса с пространством имен можно так:

```
echo MyClass::class;  // main\MyClass
```

После создания объекта в переменной сохраняется идентификатор объекта, а не сам объект. Таким образом, если присвоить значение одной переменной другой переменной, то мы не получим копию объекта. Мы получим лишь копию идентификатора, который указывает на тот же самый объект:

```
$obj = new MyClass();
// $obj2 содержит копию идентификатора, а не копию объекта!!!
$obj2 = $obj;
$obj2->x = 22;
// Переменные ссылаются на один и тот же объект!
echo $obj->toString();  // MyClass $x = 22
echo $obj2->toString(); // MyClass $x = 22
```

Чтобы создать копию объекта, нужно воспользоваться оператором `clone`:

```
$obj = new MyClass();
// $obj2 содержит копию объекта
$obj2 = clone $obj;
$obj2->x = 22;
// Переменные ссылаются на разные объекты
echo $obj->toString();  // MyClass $x = 10
echo $obj2->toString(); // MyClass $x = 22
```

При использовании оператора `clone` следует учитывать, что по умолчанию создается *поверхностная копия объекта*. Это означает, что если объект содержит в свойствах идентификаторы других объектов, то при копировании мы получим лишь

копии идентификаторов. Чтобы создать полную копию объекта, нужно внутри класса определить метод со специальным названием `__clone()` и внутри него самостоятельно создавать копии объектов (листинг 5.64).

Листинг 5.64. Создание полной копии объекта

```
class A {
   public $x = 10;                 // Свойство
}
class B {
   public $obj = null;             // Свойство
   public function __construct() { // Конструктор
      $this->obj = new A();
   }
   public function __clone() {
      $this->obj = clone $this->obj;
   }
}
$b = new B();
$c = clone $b;
$c->obj->x = 5;
echo $b->obj->x; // 10
echo $c->obj->x; // 5
```

Для удаления экземпляра класса служит оператор `unset()`:

```
unset(<Экземпляр класса>)
```

Экземпляр класса можно удалить, присвоив переменной значение `null`:

```
<Экземпляр класса> = null;
```

5.12.2. Объявление свойств внутри класса

Для создания свойства внутри класса применяется следующий синтаксис:

```
<Область видимости> <Имя переменной со знаком $>[ = <Константа>];
```

В параметре `<Область видимости>` может быть указан один из модификаторов доступа:

- `public` — открытый. Свойство доступно для внешнего использования;
- `private` — закрытый. Свойство доступно только внутри самого класса. Чаще всего свойства класса объявляются как закрытые;
- `protected` — защищенный. Свойство недоступно для внешнего использования, но доступно для самого класса и для его потомков.

Если модификатор доступа для свойства не указан, то интерпретатор выведет сообщение об ошибке.

Модификаторы доступа предназначены для контроля значения свойств класса, которые используются только для внутренней реализации класса. Например, если в свойстве предполагается хранение определенных значений, то перед присвоением значения мы можем проверить соответствие значения некоторому условию внутри общедоступного метода. Если же любой пользователь будет иметь возможность ввести что угодно, минуя нашу проверку, то ни о каком контроле не может быть и речи. Такая концепция сокрытия данных называется *инкапсуляцией*.

Присвоить или получить значение свойства после создания объекта можно через экземпляр класса с помощью оператора `->`:

```
<Переменная>-><Имя свойства без знака $> = <Значение>;
<Значение> = <Переменная>-><Имя свойства без знака $>;
```

> ***Примечание***
>
> Если свойство с указанным названием не существует в классе, то вызываются магические методы `__set()` (присваивание значения) или `__get()` (получение значения). Описание этих методов приведено в *разд. 5.12.17*.

Создадим класс `Point` с двумя общедоступными свойствами: `$x` и `$y`:

```
class Point {
   public $x = 0;
   public $y = 0;
}
```

Теперь создадим экземпляр класса `Point`, присвоим свойствам значения, а затем получим их и выведем в окно Web-браузера:

```
$obj = new Point();
$obj->x = 10;
$obj->y = 2;
var_dump($obj->x); // int(10)
var_dump($obj->y); // int(2)
```

Мы можем создать множество экземпляров класса `Point`, и каждый экземпляр будет иметь свои собственные значения свойств `$x` и `$y`, не зависящие от других экземпляров:

```
$obj = new Point();
$obj2 = new Point();
$obj->x = 10;
$obj->y = 20;
$obj2->x = 30;
$obj2->y = 40;
echo $obj->x . ' ' . $obj->y;   // 10 20
echo $obj2->x . ' ' . $obj2->y; // 30 40
```

Внутри одного класса допускается использование объектов другого класса. В качестве примера используем класс `Point` для описания координат прямоугольника внутри класса `Rectangle` (прямоугольник).

Вначале создадим класс Rectangle:

```
class Rectangle {
   public $topLeft;
   public $bottomRight;
}
```

Теперь создадим экземпляр класса Rectangle и обратимся к свойствам:

```
$obj = new Rectangle();
$obj->topLeft = new Point();
$obj->bottomRight = new Point();
$obj->topLeft->x = 0;
$obj->topLeft->y = 0;
$obj->bottomRight->x = 100;
$obj->bottomRight->y = 50;
echo $obj->topLeft->x . ' ' . $obj->topLeft->y;         // 0 0
echo $obj->bottomRight->x . ' ' . $obj->bottomRight->y; // 100 50
```

Как видно из примера, для обращения к свойствам класса Point необходимо вначале получить доступ к свойствам класса Rectangle, а затем уже с помощью оператора -> можно изменить и значение свойства класса Point:

```
$obj->bottomRight->x = 100;
```

Если свойство не существует в классе, и метод __set() отсутствует, то операция присваивания динамически создаст свойство в экземпляре класса:

```
class MyClass { }
$obj = new MyClass();
$obj->x = 5;
$obj->y = 10;
echo $obj->x;  // 5
echo $obj->y;  // 10
```

5.12.3. Определение методов внутри класса

Метод внутри класса создается так же, как и обычная функция, с помощью ключевого слова function:

```
[<Область видимости>] function <Имя метода> ([Параметры]) {
   // Тело метода
}
```

В параметре <Область видимости> указывается один из модификаторов доступа: public, private или protected. Смысл их точно такой же, что и у модификаторов свойств. Если модификатор доступа не указан, то используется модификатор public.

В предыдущем разделе мы сделали свойства класса Point общедоступными, нарушив этим один из принципов ООП. Давайте это исправим и объявим свойства

с помощью ключевого слова `private`. Чтобы иметь возможность контролировать изменение значений свойств, добавим несколько общедоступных методов (листинг 5.65).

Листинг 5.65. Класс `Point`

```
<?php
declare(strict_types=1);
namespace main;

class Point {
   private $x = 0;
   private $y = 0;
   public function setX(int $x) : void {
      $this->x = $x;
   }
   public function setY(int $y) : void {
      $this->y = $y;
   }
   public function getX() : int {
      return $this->x;
   }
   public function getY() : int {
      return $this->y;
   }
   public function __toString() : string {
      return $this->getX() . ' ' . $this->getY();
   }
}
```

Теперь обратиться к свойствам напрямую через экземпляр класса уже не получится:

```
$obj = new Point();
$obj->x = 10;        // Ошибка! Свойство объявлено как private
```

Чтобы присвоить значение полю `$x`, необходимо использовать метод `setX()`. Обращение к методам осуществляется с помощью оператора `->`:

```
<Экземпляр класса>-><Имя метода>();
```

Внутри круглых скобок передаются какие-либо значения (если метод не принимает параметры, указываются только круглые скобки):

```
$obj = new Point();
$obj->setX(100);    // OK
```

Внутри этого метода при необходимости в любой момент времени мы можем добавить код, проверяющий корректность нового значения. Таким образом соблюдается принцип сокрытия данных, называемый *инкапсуляцией*.

> **ПРИМЕЧАНИЕ**
>
> Если метод с указанным названием не существует в классе, то вызывается магический метод `__call()` (см. *разд. 5.12.17*).

Получить значение свойства `$x` позволяет метод `getX()`:

```
echo $obj->getX();  // 100
```

Общепринято использовать приставку `set` для имен методов, изменяющих значение свойства, и `get` — для имен методов, возвращающих значение свойства.

Теперь давайте посмотрим на доступ к свойствам и методам внутри класса. Для этого используется указатель `$this`:

```
$this-><Имя свойства без знака $> = <Значение>;
<Значение> = $this-><Имя свойства без знака $>;
$this-><Имя метода>();
```

Пример:

```
public function __toString() : string {
  return $this->getX() . ' ' . $this->getY();
}
```

Внутри методов класса мы имеем доступ ко всем членам класса, включая закрытые. Поэтому инструкцию `return` можно записать следующим образом:

```
return $this->x . ' ' . $this->y;
```

Метод с названием `__toString()` (перед именем два символа подчеркивания) является специальным, его еще называют *магическим*. Он будет автоматически вызван при попытке преобразования объекта в строку — например, при использовании оператора `echo`:

```
echo $obj; // 100 0
```

Если метод отсутствует, то будет выведено следующее сообщение об ошибке:

```
Recoverable fatal error: Object of class main\Point could not be
converted to string
```

5.12.4. Конструктор и деструктор

Чтобы при создании экземпляра класса присвоить начальные значения свойствам, необходимо создать метод, имеющий предопределенное название `__construct()` (перед именем два символа подчеркивания). Такой метод называется *конструктором*. Конструктор всегда автоматически вызывается при создании объекта с помощью оператора `new`.

```
public function __construct(<Параметр>) {
   $this-><Имя свойства без знака $> = <Параметр>;
}
```

При создании экземпляра класса `<Параметр>` можно указать после имени класса в круглых скобках:

```
<Экземпляр класса> = new <Имя класса>(<Параметр>);
```

Если конструктор вызывается при создании объекта, то перед уничтожением объекта автоматически вызывается метод, называемый *деструктором*. В языке PHP деструктор реализуется в виде метода с предопределенным названием `__destruct()`. Внутри этого метода обычно производится освобождение ресурсов, например закрытие ранее открытых файлов.

Пример использования конструктора и деструктора приведен в листинге 5.66.

Листинг 5.66. Конструктор и деструктор

```
<?php
declare(strict_types=1);
namespace main;

class Point {
   private $x;
   private $y;
   // Конструктор
   public function __construct(int $x = 0, int $y = 0) {
      $this->x = $x;
      $this->y = $y;
      echo "Вызван конструктор<br>\n";
   }
   // Деструктор
   public function __destruct() {
      echo "Вызван деструктор<br>\n";
   }
   public function __toString() : string {
      return $this->x . ' ' . $this->y . "<br>\n";
   }
}
```

Теперь мы можем создать объекты несколькими способами:

```
$obj = new Point();
echo $obj;
$obj2 = new Point(100, 50);
echo $obj2;
```

Результат в окне Web-браузера:

```
Вызван конструктор
0 0
Вызван конструктор
100 50
Вызван деструктор
Вызван деструктор
```

5.12.5. Наследование

Наследование — это возможность создания производных классов на основе базового класса. При этом производный класс автоматически получает возможности базового класса и может добавить новую функциональность или переопределить некоторые методы. *Базовый класс* называют также *суперклассом* или *родительским классом*, а *производный класс* — *подклассом* или *дочерним классом*. Пример создания иерархии классов приведен в листинге 5.67.

Листинг 5.67. Наследование

```
<?php
declare(strict_types=1);
namespace main;

class A {              // Базовый класс
   public function func1() { echo "A.func1()\n"; }
}
class B extends A {  // Класс B наследует класс A
   public function func2() { echo "B.func2()\n"; }
}
class C extends B {  // Класс C наследует классы A и B
   public function func3() { echo "C.func3()\n"; }
}

$obj = new C();
$obj->func1();       // A.func1()
$obj->func2();       // B.func2()
$obj->func3();       // C.func3()
```

В этом примере вначале класс `B` наследует все члены класса `A`. Затем объявляется класс `C`, который наследует все члены и класса `B`, и класса `A`. Каждый класс добавляет один новый метод. В результате экземпляр класса `C` имеет доступ ко всем методам классов `A` и `B`. Класс `A` является базовым классом (суперклассом), а класс `B` — производным классом (подклассом). В то же время класс `B` является базовым для класса `C`.

При наследовании используется следующий формат определения класса:

```
class <Производный класс> extends <Базовый класс> {
   <Описание членов класса>;
}
```

Необходимо учитывать, что наследуются только общедоступные члены (объявленные с помощью ключевого слова `public`) и защищенные члены (объявленные с помощью ключевого слова `protected`). Закрытые члены класса (объявленные с помощью ключевого слова `private`) не наследуются, и доступа к ним внутри производного класса нет.

Конструктор и деструктор в базовом классе автоматически не вызываются, если они были переопределены в производном классе. Для передачи значений конструкторам базовых классов используется следующий синтаксис:

```
parent::__construct([<Значения>]);
```

Для вызова деструктора базового класса выполняется такая инструкция:

```
parent::__destruct();
```

Продемонстрируем передачу значений конструкторам базовых классов на примере (листинг 5.68).

Листинг 5.68. Передача значений конструкторам базовых классов

```
<?php
declare(strict_types=1);
namespace main;

class A {
   private $x;
   public function __construct($x) {          // Конструктор
      $this->x = $x;
      echo "A.__construct(); x = {$this->x}<br>\n";
   }
}
class B extends A {
   private $y;
   public function __construct($x, $y) {      // Конструктор
      parent::__construct($x);
      $this->y = $y;
      echo "B.__construct(); y = {$this->y}<br>\n";
   }
}
class C extends B {
   private $z;
   public function __construct($x, $y, $z) {  // Конструктор
      parent::__construct($x, $y);
      $this->z = $z;
      echo "C.__construct(); z = {$this->z}<br>\n";
   }
}
$obj = new C(1, 2, 3);
```

Результат выполнения программы:

```
A.__construct(); x = 1
B.__construct(); y = 2
C.__construct(); z = 3
```

Примечание

Язык PHP не поддерживает множественное наследование.

5.12.6. Переопределение методов базового класса

Чтобы переопределить метод базового класса, достаточно в производном классе создать одноименный метод. В этом случае будет вызван метод только из производного класса. Чтобы вызвать метод из базового класса внутри производного класса, используется следующий синтаксис:

```
parent::<Метод>([<Значения>])
```

Рассмотрим переопределение метода на примере (листинг 5.69).

Листинг 5.69. Переопределение метода базового класса

```
<?php
declare(strict_types=1);
namespace main;

class A {
   public function func() { echo "A.func()<br>\n"; }
}
class B extends A {
   public function func() {
      echo "B.func()<br>\n";
      parent::func(); // Вызываем метод базового класса
   }
}
$obj = new B();
$obj->func();
```

Результат в окне Web-браузера:

```
B.func()
A.func()
```

Внимание!

Конструктор и деструктор в родительском классе автоматически не вызываются. Для их вызова также необходимо использовать ключевое слово `parent`.

5.12.7. Финальные классы и методы

В ряде случаев может возникнуть необходимость предотвратить наследование всего класса или запретить переопределение отдельных методов. Для этого используется ключевое слово `final`. Чтобы запретить наследование всего класса, ключевое слово `final` указывается перед ключевым словом `class`:

```
final class A {
   public function func() { echo "A.func()<br>\n"; }
}
```

При запрете переопределения метода ключевое слово `final` указывается так:

```
class A {
   final public function func() { echo "A.func()<br>\n"; }
}
```

> **Примечание**
>
> Свойства не могут быть объявлены финальными.

5.12.8. Абстрактные классы и методы

Абстрактные методы содержат только объявление метода — без реализации. Создать экземпляр класса, в котором объявлен абстрактный метод, нельзя. Предполагается, что производный класс должен переопределить метод и реализовать его функциональность. Абстрактные методы объявляются с помощью ключевого слова `abstract` по следующей схеме:

```
abstract [<Модификатор>] function <Имя метода>([<Параметры>]);
```

Обратите внимание: метод не имеет блока. Сразу после списка параметров за закрывающей круглой скобкой ставится точка с запятой. Если внутри класса существует хотя бы один абстрактный метод, то весь класс следует объявить абстрактным, добавив перед ключевым словом `class` ключевое слово `abstract`. Следует учитывать, что абстрактный класс может помимо абстрактных содержать и обычные методы. Кроме того, класс может быть объявлен абстрактным, даже если он не содержит абстрактных методов. Пример объявления и замещения абстрактного метода приведен в листинге 5.70.

Листинг 5.70. Абстрактные классы и методы

```
<?php
declare(strict_types=1);
namespace main;

abstract class A {
   abstract protected function func();  // Абстрактный метод
   public function test() {             // Обычный метод
      echo "A.test()\n";
   }
}
class B extends A {
   public function func() {             // Замещаем метод
      echo "B.func()\n";
   }
}
```

```
$obj = new B();
$obj->func(); // B.func()
$obj->test(); // A.test()
```

Обратите внимание: при замещении метода в производном классе ключевое слово `abstract` не указывается. Кроме того, область видимости замещенного метода должна быть такой же или менее строгой. Например, если указан модификатор доступа `protected`, то можно указать или `protected`, или `public`, но не `private`. Тип данных и число параметров у замещающего метода должно совпадать с типом данных и числом параметров в методе базового класса.

5.12.9. Объявление констант внутри класса

Константа внутри класса объявляется с помощью ключевого слова `const`:

```
[<Область видимости>] const <Имя константы без $> = <Значение>;
```

Если область видимости не указана, то используется модификатор доступа `public`. Обратите внимание: модификаторы доступа для констант введены в версии 7.1 — ранее область видимости не указывалась, и все константы были общедоступными.

Доступ к константе вне класса осуществляется без создания экземпляра класса, т. к. константа существует в единственном экземпляре, а не отдельно для каждого объекта:

```
<Имя класса>::<Имя константы без $>
```

Внутри класса к константе можно обратиться через имя класса или с помощью ключевого слова `self`:

```
<Имя класса>::<Имя константы без $>
self::<Имя константы без $>
```

Пример использования констант приведен в листинге 5.71.

Листинг 5.71. Пример использования констант

```
<?php
declare(strict_types=1);
namespace main;

class MyClass {
   public const MY_CONST = 10;
   private const PI = 3.14;
   public function getPI() {
      return self::PI;
   }
}
$obj = new MyClass();
echo MyClass::MY_CONST; // 10
```

```
// echo MyClass::PI;    // Ошибка! Константа private
echo $obj->getPI();     // 3.14
```

5.12.10. Статические свойства и методы

Внутри класса можно создать свойство или метод, которые будут доступны без создания экземпляра класса. Для этого перед определением свойства или метода следует указать ключевое слово `static`:

```
[<Область видимости>] static <Имя свойства> = <Константа>;
[<Область видимости>] static function <Имя метода> ([Параметры]) {
   // Тело метода
}
```

Статические члены класса существуют в единственном экземпляре, независимо от количества созданных объектов. Обратите внимание на то, что внутри статического метода невозможно получить доступ к обычным свойствам — только к статическим членам класса. Если модификатор доступа не указан, то статические члены классы являются общедоступными.

Доступ к статическим свойствам и методам вне класса осуществляется так:

```
<Название класса>::<Имя свойства со знаком $>
<Название класса>::<Имя метода>([<Параметры>])
```

Способы обращения к статическим членам внутри класса:

```
<Название класса>::<Имя свойства со знаком $>
<Название класса>::<Имя метода>([<Параметры>])
self::<Имя свойства со знаком $>
self::<Имя метода>([<Параметры>])
```

Различные варианты создания статических членов класса и способы доступа к ним показаны в листинге 5.72.

Листинг 5.72. Статические свойства и методы

```
<?php
declare(strict_types=1);
namespace main;

class MyClass {
   public static $x = 10;
   public static function getX() {
      return self::$x;
   }
}
echo MyClass::$x;       // 10
MyClass::$x = 22;
echo MyClass::getX();  // 22
```

```
$c = 'main\MyClass';   // Доступ через переменную
$c::$x = 33;
echo $c::getX();       // 33
```

> **Примечание**
>
> Если статический метод с указанным названием не существует в классе, то вызывается магический метод `__callStatic()` (см. *разд. 5.12.17*).

5.12.11. Методы-фабрики

Как вы уже знаете, при создании экземпляра класса указывается оператор `new`, после которого следует название класса и круглые скобки, внутри которых передаются значения конструктору класса. Можно также создать статический метод, который будет использоваться для создания объектов. Название этого метода может быть произвольным. Такие методы принято называть *методами-фабриками* (листинг 5.73).

Листинг 5.73. Методы-фабрики

```
<?php
declare(strict_types=1);
namespace main;

class MyClass {
   private $x;
   private function __construct($x) {
      $this->x = $x;
   }
   // Статический метод-фабрика
   public static function getInstance($x) {
      return new MyClass($x);
   }
   public function getX() {
      return $this->x;
   }
}
// Так нельзя, у конструктора модификатор private
// $obj1 = new MyClass(10);
$obj2 = MyClass::getInstance(10);
$obj3 = MyClass::getInstance(45);
echo $obj2->getX();  // 10
echo $obj3->getX();  // 45
```

5.12.12. Полиморфизм

В результате наследования производный класс получает всю функциональность базового класса и может переопределить некоторые методы. После переопределе-

ния метода через экземпляр класса будет вызван метод производного класса — даже в том случае, если вызов метода производится внутри базового класса (листинг 5.74). Такое поведение называется *полиморфизмом* — смысл действия, которое выполняет одноименный метод, зависит от объекта, над которым это действие выполняется.

Листинг 5.74. Полиморфизм

```
class A {
   public function func() { echo "A.func()\n"; }
   public function test() { $this->func(); }
}
class B extends A {
   public function func() { echo "B.func()\n"; }
}
$obj = new A();
$obj->test();    // A.func()
$obj = new B();
$obj->test();    // B.func()
```

5.12.13. Позднее статическое связывание

Если при наследовании переопределяются статические методы, то ключевое слово `self` будет указывать на метод базового класса, а не на одноименный метод производного класса. При использовании статических методов динамическое связывание не работает. Чтобы получить результат как при полиморфизме, следует использовать *позднее статическое связывание* (листинг 5.75). В этом случае вместо ключевого слова `self` нужно указать ключевое слово `static`.

Листинг 5.75. Позднее статическое связывание

```
class A {
   public static function func() { echo "A.func()\n"; }
   public static function test() { self::func(); }
   // Позднее статическое связывание
   public static function test2() { static::func(); }
}
class B extends A {
   public static function func() { echo "B.func()\n"; }
}
B::test();  // A.func()
B::test2(); // B.func()
```

5.12.14. Передача объектов в функцию

Если функция принимает объект, то класс объекта можно указать в качестве типа данных в объявлении параметра. Следует помнить, что при вызове функции и

передаче объекта параметр получит копию значения, а значением является идентификатор объекта. Свойства этого объекта можно будет изменить внутри функции через эту копию идентификатора.

Если метод объявлен внутри класса с тем же именем, что и класс объекта, то внутри метода можно будет обратиться ко всем членам класса, включая закрытые. Если метод объявлен в другом классе, то к закрытым методам доступа не будет (листинг 5.76).

Листинг 5.76. Передача объектов в функцию

```
<?php
declare(strict_types=1);
namespace main;

class A {
   private $x = 10;
   public $y = 5;
   public static function test(A $obj) {
      echo $obj->x, "\n"; // OK, хотя $x — private
   }
}
class B {
   public static function test(A $obj) {
      // Нельзя, $x — private и класс не A
      // echo $obj->x, "\n";
   }
}
function test(A $obj) {
   $obj->y = 11;           // Можно изменить объект!
   echo $obj->y, "\n";
}

$c = new A();
test($c);              // 11
// Свойство объекта изменилось!
echo $c->y, "\n";     // 11
A::test($c);           // 10
B::test($c);           // Ошибка
```

Для передачи иерархии классов и использования полиморфизма в качестве типа параметра указывается тип базового класса. В этом случае будет вызван одноименный метод из производного класса. Помните, что базовый класс должен содержать все необходимые методы, т. к. способа приведения объекта одного класса к другому в PHP нет. Пример указания базового класса в качестве типа параметра приведен в листинге 5.77.

Листинг 5.77. Указание базового класса в качестве типа параметра

```
<?php
declare(strict_types=1);
namespace main;

abstract class A {
   abstract public function func1();
   abstract public function func2();
}
class B extends A {
   public function func1() { echo "B.func1() "; }
   public function func2() { echo "B.func2() "; }
}
class C extends A {
   public function func1() { echo "C.func1() "; }
   public function func2() { echo "C.func2() "; }
}
class D { // Класс D не наследует класс A
   public function func1() { echo "D.func1() "; }
   public function func2() { echo "D.func2() "; }
}
function test(A $obj) { // Указан базовый класс
   $obj->func1();
   $obj->func2();
}
$b = new B();
$c = new C();
// Можем передать в функцию оба объекта,
// т. к. они наследники класса A
test($b);               // B.func1() B.func2()
test($c);               // C.func1() C.func2()
$d = new D();
if ($d instanceof A) { // Не наследник класса A
   test($d);
}
else echo 'Не наследник класса A';
```

Так как в функцию передается значение переменной, то можно сказать, что в функцию всегда передается копия идентификатора, а значит, внутри функции мы можем изменить состояние объекта:

```
class MyClass { public $x = 10; }
function test(MyClass $c) {
   // Внутри функции можно изменить свойства объекта!
   $c->x = 22;
}
```

```
$obj = new MyClass();
// Передается копия идентификатора, а не копия объекта!
test($obj);
echo $obj->x;  // 22
```

Следует учитывать, что передача копии идентификатора объекта и передача параметра по ссылке — это не одно и то же. В первом случае мы можем изменить состояние объекта через копию идентификатора, но не можем присвоить глобальной переменной ссылку на другой объект:

```
class MyClass { public $x = 10; }
function test(MyClass $c) { // Передача копии идентификатора
   $c = new MyClass();
   $c->x = 22;
   // Значение $obj НЕ изменится!
}
$obj = new MyClass();
test($obj);
echo $obj->x;  // 10
```

Во втором случае можно изменить значение глобальной переменной, т. к. ссылка ведет на переменную, а не на объект:

```
class MyClass { public $x = 10; }
function test(MyClass &$c) { // Передача параметра по ссылке
   $c = new MyClass();
   $c->x = 22;
   // Значение $obj изменится!
}
$obj = new MyClass();
test($obj);
echo $obj->x;  // 22
```

Если внутри метода мы создаем анонимную функцию, то внутри тела анонимной функции будут видны указатели `$this` и `self`:

```
class MyClass {
   private $x = 10;
   private static $y = 5;
   public function test() {
      return function () { return $this->x . ' ' . self::$y; };
   }
}
$obj = new MyClass();
$func = $obj->test();
echo $func();  // 10 5
```

5.12.15. Оператор *instanceof*

В листинге 5.77 для проверки типа мы воспользовались оператором `instanceof`. Формат оператора:

```
<Объект> instanceof <Имя класса или интерфейса>
```

Оператор `instanceof` позволяет проверить:

- является ли объект экземпляром указанного класса;
- является ли объект экземпляром класса, наследующего указанный класс;
- реализует ли класс объекта указанный интерфейс.

Если условия соблюдаются, то оператор возвращает значение `true`, в противном случае — значение `false`.

5.12.16. Приведение типов

Число, строку или массив можно преобразовать в объект. Достаточно выполнить приведение к типу `object`:

```
$x = 10;
$y = 1.2;
$s = 'Строка';
$arr = [1, 2, 3];
var_dump((object)$x);
// object(stdClass)#1 (1) { ["scalar"]=> int(10) }
var_dump((object)$y);
// object(stdClass)#2 (1) { ["scalar"]=> float(1.2) }
var_dump((object)$s);
// object(stdClass)#3 (1) { ["scalar"]=> string(12) "Строка" }
var_dump((object)$arr);
// object(stdClass)#4 (3) { [0]=> int(1) [1]=> int(2) [2]=> int(3) }
```

В этом случае возвращается экземпляр класса `stdClass`. Значение доступно через свойство `scalar`:

```
$obj = (object)5;
echo $obj->scalar; // 5
```

В случае со списком названиями свойств станут числа, и получить их значения можно только с помощью механизма итераций:

```
$obj = (object)[1, 2, 3];
foreach ($obj as $value) {
   echo $value . ' ';
} // 1 2 3
```

Попытка преобразования объекта в типы `int` и `float` закончится ошибкой. Объект можно привести к массиву. В этом случае ключами станут имена свойств. Если внутри класса реализован метод с предопределенным названием `__toString()`, то объект можно будет преобразовать в строку (листинг 5.78).

Листинг 5.78. Преобразование объекта в строку и массив

```
class MyClass {
   public $x = 10;
   public $y = 1.2;
   public function __toString() { // Преобразование в строку
      return $this->x . ' ' . $this->y;
   }
}
$obj = new MyClass();
var_dump((string)$obj);
// string(6) "10 1.2"
var_dump((array)$obj);
// array(2) { ["x"]=> int(10) ["y"]=> float(1.2) }
```

5.12.17. Магические методы

В предыдущих разделах мы познакомились с методами `__construct()` (конструктор класса), `__destruct()` (деструктор класса), `__clone()` (клонирование объектов) и `__toString()` (преобразование объекта в строку), которые называются *магическими*, т. к. явным образом мы их не вызываем. Эти методы вызываются автоматически в определенных ситуациях — например, при любой попытке использовать объект в строковом контексте будет вызываться метод `__toString()`. Помимо рассмотренных методов существует и множество других — полный их список смотрите в документации.

Метод `__invoke()` вызывается, когда после имени переменной указаны круглые скобки. Внутри круглых скобок можно передать методу какие-либо значения (листинг 5.79).

Листинг 5.79. Метод `__invoke()`

```
class MyClass {
   public $x = 10;
   public function __invoke($x) {
      $this->x = $x;
      echo "Вызван метод __invoke()\n";
   }
}
$obj = new MyClass();
$obj(5);       // Вызывается __invoke()
echo $obj->x; // 5
```

Метод `__debugInfo()` вызывается при использовании функций `print_r()` и `var_dump()`. Внутри метода нужно вернуть ассоциативный массив с именами и значениями свойств (листинг 5.80).

Листинг 5.80. Метод `__debugInfo()`

```
class MyClass {
   public $x = 10;
   public function __debugInfo() {
      echo "Вызван метод __debugInfo()\n";
      return ['x' => $this->x];
   }
}
$obj = new MyClass();
var_dump($obj);  // Вызывается __debugInfo()
print_r($obj);   // Вызывается __debugInfo()
```

Метод `__call()` вызывается при отсутствии обычного метода в классе, а статический метод `__callStatic()` — при отсутствии статического метода. В первом параметре методы принимают название вызванного метода, а во втором — список с переданными значениями. Пример использования методов `__call()` и `__callStatic()` приведен в листинге 5.81.

Листинг 5.81. Методы `__call()` и `__callStatic()`

```
class MyClass {
   public function __call($name, $args) {
      echo "Вызван метод __call()\n";
      var_dump($name);
      var_dump($args);
   }
   public static function __callStatic($name, $args) {
      echo "Вызван метод __callStatic()\n";
      var_dump($name);
      var_dump($args);
   }
}
$obj = new MyClass();
$obj->test(1);      // Вызывается __call()
MyClass::func(5);   // Вызывается __callStatic()
```

Метод `__set()` вызывается при попытке присвоить значение отсутствующему свойству, а метод `__get()` — при попытке получить значение отсутствующего свойства. В первом параметре методы принимают название свойства. Новое значение доступно через второй параметр метода `__set()`. Пример использования методов `__set()` и `__get()` приведен в листинге 5.82.

Листинг 5.82. Методы `__set()` и `__get()`

```
class MyClass {
   public function __set($name, $value) {
      echo "Вызван метод __set()\n";
```

```
      var_dump($name);
      var_dump($value);
   }
   public function __get($name) {
      echo "Вызван метод __get()\n";
      var_dump($name);
      return 15;
   }
}
$obj = new MyClass();
$obj->x = 3;          // Вызывается __set()
$x = $obj->x;         // Вызывается __get()
echo $x;              // 15
```

Метод `__unset()` вызывается при попытке вызова функции `unset()` для отсутствующего свойства, а метод `__isset()` — при попытке вызова функций `isset()` и `empty()` для отсутствующего свойства. В качестве параметра методы принимают название свойства. Внутри метода `__isset()` нужно вернуть логическое значение. Пример использования методов `__unset()` и `__isset()` приведен в листинге 5.83.

Листинг 5.83. Методы `__unset()` и `__isset()`

```
class MyClass {
   public function __unset($name) {
      echo "Вызван метод __unset()\n";
      var_dump($name);
   }
   public function __isset($name) {
      echo "Вызван метод __isset()\n";
      var_dump($name);
      return true;
   }
}
$obj = new MyClass();
isset($obj->x);   // Вызывается __isset()
empty($obj->x);   // Вызывается __isset()
unset($obj->x);   // Вызывается __unset()
```

> ***Примечание***
> Описание магических методов `__sleep()` и `__wakeup()` см. в *разд. 5.13.8*.

5.12.18. Сравнение объектов

Сравнить два объекта можно с помощью операторов == и ===. В первом случае объекты будут равны, если они содержат одинаковые свойства с одинаковыми значе-

ниями и являются экземплярами одного и того же класса. Во втором случае сравниваются идентификаторы объектов — объекты будут равны, если обе переменные ссылаются на один и тот же объект.

Пример сравнения объектов приведен в листинге 5.84.

Листинг 5.84. Сравнение объектов

```
class A {
   public $x = 10;
}
class B {
   public $x = 10;
}
$a1 = new A();
$a2 = new A();
$b = new B();
var_dump($a1 == $a2);   // bool(true)
var_dump($a1 === $a2);  // bool(false)
var_dump($a1 == $b);    // bool(false)
```

5.12.19. Автоматическая загрузка классов

Часто классы размещают в отдельных файлах с названием, совпадающим с названием класса. При этом подключение всех файлов может занять много места. С помощью функции `spl_autoload_register()` можно зарегистрировать обработчик, в который будет передаваться название класса (с предваряющим названием пространства имен), требующего загрузки. Внутри обработчика нужно выполнить подключение файла с классом — например, с помощью оператора `include`.

Давайте в каталоге C:\xampp\php\includes создадим каталог main (название нашего пространства имен), в который добавим файлы Class1.inc (листинг 5.85) и Class2.inc (листинг 5.86).

Листинг 5.85. Содержимое файла C:\xampp\php\includes\main\Class1.inc

```
<?php
namespace main;
class Class1 {
   public $x = 10;
}
?>
```

Листинг 5.86. Содержимое файла C:\xampp\php\includes\main\Class2.inc

```
<?php
namespace main;
```

```
class Class2 {
   public $x = 5;
}
?>
```

Теперь выполним автоматическую загрузку этих файлов с классами в основной программе (листинг 5.87).

Листинг 5.87. Автоматическая загрузка классов

```
<?php
namespace main;
spl_autoload_register(function ($name) {
   @include $name . '.inc';
});

$c1 = new Class1();
$c2 = new Class2();
echo $c1->x; // 10
echo $c2->x; // 5
```

Обратите внимание: параметр `$name` внутри обработчика будет содержать не только название класса, но и название пространства имен, например `main\Class1`. Именно ради этого мы и разместили файлы с классами внутри каталога main, название которого совпадает с названием пространства имен.

5.12.20. Функции для работы с классами и объектами

Приведем функции, предназначенные для работы с классами и объектами:

- `get_class([<Объект>])` — возвращает название класса указанного объекта с предваряющим названием пространства имен. Если параметр не указан, то возвращает название класса, внутри которого функция вызвана:

  ```
  class MyClass {}
  $obj = new MyClass();
  echo get_class($obj); // main\MyClass
  echo MyClass::class;  // main\MyClass
  ```

- `get_parent_class([<Объект>])` — возвращает название родительского класса для указанного объекта с предваряющим названием пространства имен. Если параметр не указан, и функция вызывается внутри класса, то возвращает название родительского класса для текущего класса. Если родительского класса нет, то функция возвращает значение `false`;
- `get_declared_classes()` — возвращает список всех доступных сценарию классов (и пользовательских и встроенных):

  ```
  print_r(get_declared_classes());
  ```

- `class_exists()` — возвращает значение `true`, если указанный класс объявлен, и `false` — в противном случае. Название класса указывается с предваряющим названием пространства имен. Формат функции:

  ```
  class_exists(string $name[, bool $autoload=true]) : bool
  ```

  Пример:

  ```
  var_dump(class_exists('main\\MyClass'));  // bool(true)
  ```

- `class_alias()` — создает псевдоним для класса:

  ```
  class_alias('main\\MyClass', 'main\\MyClass2');
  var_dump(class_exists('main\\MyClass2')); // bool(true)
  ```

- `get_object_vars(<Объект>)` — возвращает массив с названиями и значениями доступных свойств для указанного объекта. Вне класса будут видны только общедоступные свойства, а внутри класса — все свойства класса:

  ```
  $obj = new MyClass();
  print_r(get_object_vars($obj));
  ```

- `get_class_vars(<Класс>)` — возвращает массив с названиями и начальными значениями доступных свойств для указанного класса. Вне класса будут видны только общедоступные свойства, а внутри класса — все свойства класса:

  ```
  print_r(get_class_vars('main\\MyClass'));
  ```

- `get_class_methods(<Класс>)` — возвращает список с названиями доступных методов для указанного класса. Вне класса будут видны только общедоступные методы, а внутри класса — все методы класса:

  ```
  print_r(get_class_methods('main\\MyClass'));
  ```

- `method_exists()` — возвращает значение `true`, если указанный во втором параметре метод существует в классе, и `false` — в противном случае. Формат функции:

  ```
  method_exists(mixed $object, string $method_name) : bool
  ```

  Пример:

  ```
  $obj = new MyClass();
  var_dump(method_exists($obj, 'func'));
  var_dump(method_exists('main\\MyClass', 'func'));
  ```

- `property_exists()` — возвращает значение `true`, если указанное во втором параметре свойство существует в классе, и `false` — в противном случае. Формат функции:

  ```
  property_exists(mixed $class, string $property) : bool
  ```

  Пример:

  ```
  $obj = new MyClass();
  var_dump(property_exists($obj, 'x'));
  var_dump(property_exists('main\\MyClass', 'x'));
  ```

5.12.21. Создание шаблона сайта при помощи класса

При создании больших сайтов страницу обычно делят на три части: верхний колонтитул, тело страницы и нижний колонтитул. Для подключения колонтитулов к основному документу используются операторы `require` и `include`.

Нижний колонтитул практически всегда одинаков для всех страниц, а вот верхние колонтитулы по определению совпадать не могут, — для всех страниц сайта недопустим одинаковый заголовок (тег `<title>`). Более того, каждая страница должна иметь уникальное описание для поисковых роботов (тег `<meta>`).

Для реализации верхнего колонтитула создадим класс, позволяющий менять заголовок и описание страницы. Для этого сформируем три файла:

- HTML5Header.inc — верхний колонтитул (листинг 5.88);
- footer.inc — нижний колонтитул (листинг 5.89);
- index.php — основное содержание страницы (листинг 5.90).

Листинг 5.88. Содержимое файла C:\xampp\php\includes\HTML5Header.inc

```
<?php
class HTML5Header {
   private $title;
   private $description;
   private $meta;
   public function __construct($title, $description='', $meta='') {
      $this->title = $title;
      $this->description = $description;
      $this->meta = $meta;
   }
   public function show() {
      echo <<<LABEL
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>{$this->title}</title>
   <meta name="description" content="{$this->description}">
{$this->meta}
</head>
<body>

LABEL;
   }
}
?>
```

Листинг 5.89. Содержимое файла C:\xampp\php\includes\footer.inc

```
<footer>
   <p>&copy; <?=date('Y')?> г. Все права защищены</p>
</footer>
</body>
</html>
```

Листинг 5.90. Содержимое файла index.php

```
<?php
require_once 'HTML5Header.inc';
$title = 'Заголовок';
$description = 'Описание';
$meta = '   <style> body { background: #e8e8e8; } </style>';
$obj = new HTML5Header($title, $description, $meta);
$obj->show();
?>
<header>
   <h1>Заголовок страницы</h1>
</header>
<section>
   <h2>Заголовок содержимого</h2>
   <p>Основное содержание страницы</p>
</section>
<?php
require_once 'footer.inc';
?>
```

Если открыть файл index.php в Web-браузере и отобразить исходный код, то мы увидим следующий HTML-код:

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Заголовок</title>
   <meta name="description" content="Описание">
   <style> body { background: #e8e8e8; } </style>
</head>
<body>
<header>
   <h1>Заголовок страницы</h1>
</header>
<section>
   <h2>Заголовок содержимого</h2>
   <p>Основное содержание страницы</p>
</section>
```

```
<footer>
   <p>&copy; 2018 г. Все права защищены</p>
</footer>
</body>
</html>
```

Таким образом, меняя значения переменных `$title` и `$description`, можно сделать уникальными заголовок и описание для каждой страницы.

5.13. Интерфейсы

Интерфейс похож на абстрактный класс и также содержит только сигнатуру методов без реализации. Однако интерфейс не является классом — он лишь требование к реализации класса. Класс, реализующий интерфейс, гарантирует, что внутри него определены методы, которые описаны внутри блока интерфейса. Если класс не переопределяет хотя бы один метод, то он становится абстрактным классом и создать экземпляр этого класса будет нельзя.

Опишем основные различия между интерфейсами и классами:

- интерфейс не может содержать свойств;
- интерфейс может содержать константы. Переопределить константу нельзя;
- все методы, объявленные в интерфейсе, являются абстрактными. Внутри блока интерфейса указывается только сигнатура методов без реализации. Эти методы обязательно должны быть переопределены в классе, который реализует интерфейс. Внутри интерфейса можно объявить сигнатуру конструктора;
- все члены интерфейса должны иметь область видимости `public`;
- класс может реализовывать сразу несколько интерфейсов;
- интерфейсы поддерживают множественное наследование.

Зачем нужны интерфейсы? Во-первых, с помощью интерфейсов можно имитировать *множественное наследование*. Дело в том, что после ключевого слова `extends` в инструкции `class` допускается указание только одного класса, — следовательно, классы не поддерживают множественное наследование. Однако классы могут реализовывать сразу несколько интерфейсов. Кроме того, сами интерфейсы поддерживают множественное наследование.

Во-вторых, с помощью интерфейсов можно создать аналог функциям обратного вызова. В этом случае в функцию передается ссылка на объект, реализующий какой-либо интерфейс. Внутри блока интерфейса указывается сигнатура метода, который будет вызван при наступлении события. Нам достаточно в своем классе реализовать интерфейс и переопределить метод, указанный в интерфейсе. Далее создаем экземпляр класса и передаем его в метод регистрации обработчика события. При наступлении события достаточно вызвать метод из интерфейса.

В-третьих, интерфейсы полезны при командной работе над проектом. Например, команда выполняет три задачи. Первый программист реализует ввод данных, вто-

рой — их обработку, а третий — вывод обработанных данных. Если эти программисты не договорятся заранее, то они не смогут состыковать свои классы. Поэтому второй программист говорит первому: «Реализуй интерфейс с методом `read()`», а третьему: «Реализуй интерфейс с методом `write()`, названия интерфейсов такие-то, сигнатура методов такая-то». Все! Теперь каждый программист занимается своей работой. Первый программист создает множество классов (например, получение данных из строки, получение данных из Интернета и т. д.) и в каждом классе реализует интерфейс получения данных (метод `read()`). Третий программист создает множество классов для вывода результата (например, вывод в окно Web-браузера, вывод к файл, передача по сети Интернет, печать на принтере и т. д.) и в каждом классе реализует интерфейс вывода (метод `write()`). Второй программист может подключить эти классы к своим классам, указав в качестве типов параметров не названия классов, а названия интерфейсов. Поэтому не важно, как называются классы, — главное, чтобы они соответствовали требованиям интерфейса. Давайте рассмотрим это на примере.

5.13.1. Создание интерфейса

Описание интерфейса начинается с ключевого слова `interface`:

```
interface <Имя интерфейса> {
   // Объявление констант и описание сигнатур методов
}
```

Это почти как при создании класса. Только вместо ключевого слова `class` используется ключевое слово `interface`. Но интерфейс не является классом — он всего лишь содержит требования, предъявляемые к классу, его реализующему. Требования выражаются в сигнатурах методов, которые должен переопределить класс. В нашем случае давайте запишем требование реализовать метод `read()` в интерфейсе с названием `IRead` (листинг 5.91). Объявление интерфейса вынесем в отдельный файл IRead.inc, и так как будем работать с пространством имен `main`, создадим этот файл в одноименном каталоге (C:\xampp\php\includes\main\IRead.inc).

Листинг 5.91. Содержимое файла C:\xampp\php\includes\main\IRead.inc

```
<?php
namespace main;
interface IRead {
   public function read() : string;
}
?>
```

Как видно из примера, мы указали только тип возвращаемого значения и название метода. После описания метода ставится точка с запятой. Обратите внимание: метод не содержит блока с реализацией. Класс должен переопределить этот метод и выполнить его реализацию. Какой будет эта реализация, каждый класс решает сам, — главное, чтобы метод вернул строку с данными.

Теперь аналогично создадим еще один файл с интерфейсом `IWrite` и методом `write()` (листинг 5.92).

Листинг 5.92. Содержимое файла C:\xampp\php\includes\main\IWrite.inc

```
<?php
namespace main;
interface IWrite {
   public function write(string $s) : void;
}
?>
```

Как видно из сигнатуры, метод `write()` принимает строку и ничего не возвращает.

Итак, у нас есть требования к классам ввода и вывода. Теперь давайте создадим классы, реализующие эти интерфейсы.

5.13.2. Реализация интерфейса

Чтобы указать, что класс реализует какой-либо интерфейс, после названия класса (и выражения наследования, если оно есть) указывается ключевое слово `implements` и название интерфейса:

```
class <Имя класса> implements <Имя интерфейса> {
}
```

Реализация интерфейса заключается в переопределении метода из интерфейса внутри класса. Реализуем интерфейс `IRead` и внутри метода `read()` просто вернем какую-либо строку:

```
class A implements IRead {
   public function read(): string {
      return 'строка 1';
   }
}
```

Этим мы говорим, что класс `A` реализует интерфейс `IRead`.

Если класс наследует какой-либо другой класс, то ключевое слово `implements` указывается после названия базового класса:

```
class MyClass extends A implements IRead {
}
```

Класс `MyClass` наследует класс `A` и реализует интерфейс `IRead`.

Давайте создадим еще один класс, в котором вернем другую строку, а также определим какой-либо метод, например конструктор, и закрытое свойство:

```
class B implements IRead {
   private $s = '';
```

```
   public function __construct(string $s) {
      $this->s = $s;
   }
   public function read(): string {
      return $this->s;
   }
}
```

Реализация классов для ввода данных у нас есть. Мы упростили классы, но в реальной жизни класс `A` мог бы получать данные из файла, а класс `B` — из Интернета. Причем, эти классы могли бы иметь множество других методов и свойств, как и обычные классы.

Теперь создадим класс `C`, реализующий интерфейс `IWrite`:

```
class C implements IWrite {
   public function write(string $s): void {
      echo $s;
   }
}
```

В этом случае мы также поступаем упрощенно — всего лишь выводим строку в окно Web-браузера. Нам сейчас главное понять суть интерфейсов, а не разбираться с реализацией ввода и вывода.

Теперь создадим класс `D`, внутри которого будет производиться обработка данных (мы просто изменим регистр символов в строке), полученных из разных источников с помощью интерфейса `IRead`, и вывод данных согласно интерфейсу `IWrite`. Чтобы сложилась целостная картина, приведем полное содержание файла index.php (листинг 5.93). Интерфейсы у нас расположены в отдельных файлах с названиями, соответствующими именам интерфейсов. Эти файлы мы будем подключать автоматически с помощью функции `spl_autoload_register()`.

Листинг 5.93. Реализация интерфейсов. Содержимое файла index.php

```
<?php
declare(strict_types=1);
namespace main;

spl_autoload_register(function ($name) { // Автозагрузка
   @include $name . '.inc';
});

class A implements IRead {
   public function read(): string {
      return 'строка 1';
   }
}
class B implements IRead {
   private $s = '';
```

```
    public function __construct(string $s) {
        $this->s = $s;
    }
    public function read(): string {
        return $this->s;
    }
}
class C implements IWrite {
    public function write(string $s): void {
        echo $s;
    }
}
class D {
    private $ir;
    private $iw;
    private $str = '';

    public function __construct(IRead $r, IWrite $w) {
        $this->ir = $r;
        $this->iw = $w;
        $this->str = $this->ir->read();        // Получаем данные
    }
    public function change() : void {
        $this->str = mb_strtoupper($this->str,
                                   'UTF-8');   // Обрабатываем данные
    }
    public function write() : void {
        $this->iw->write($this->str);          // Выводим данные
    }
}
$obj1 = new D(new A(), new C());
$obj1->change();
$obj1->write();                                // СТРОКА 1
$obj2 = new D(new B('строка 2'), new C());
$obj2->change();
$obj2->write();                                // СТРОКА 2
```

Обратите внимание на типы данных параметров конструктора класса `D` — там нет классов `A`, `B` и `C`. Вместо этих классов указываются названия интерфейсов. Нам не важно, как называются классы ввода и вывода, нам важно, чтобы эти классы реализовывали интерфейсы ввода или вывода. Если типом параметра будет название интерфейса, то через переменную будут доступны только методы, объявленные внутри блока интерфейса. А нам ведь больше ничего и не надо — требуется лишь получить исходные данные и вывести обработанные. Никакой другой информации о реализации классов ввода и вывода нам знать не нужно — и, как можно видеть, без этой информации мы прекрасно справились с задачей.

5.13.3. Реализация нескольких интерфейсов

Один класс может реализовывать сразу несколько интерфейсов. В этом случае названия интерфейсов указываются после ключевого слова `implements` через запятую. Создадим класс `E`, который реализует и интерфейс ввода, и интерфейс вывода (листинг 5.94).

Листинг 5.94. Реализация нескольких интерфейсов

```
class E implements IRead, IWrite {
   public function read(): string {
      return 'строка 3';
   }
   public function write(string $s): void {
      echo $s;
   }
}
```

Теперь используем этот класс в нашем процессе ввода, обработки и вывода:

```
$objE = new E();
$obj3 = new D($objE, $objE);
$obj3->change();
$obj3->write();                              // СТРОКА 3
```

В этом случае мы просто передали один объект класса `E` и для операции ввода, и для операции вывода.

5.13.4. Расширение интерфейсов

Интерфейсы могут расширяться за счет наследования базовых интерфейсов. Причем интерфейсы поддерживают множественное наследование, в отличие от классов. Чтобы наследовать интерфейс, необходимо указать ключевое слово `extends` и название базового интерфейса. Если используется множественное наследование, то базовые интерфейсы записываются через запятую.

Создадим интерфейс `IReadWrite`, который наследует интерфейсы `IRead` и `IWrite`, и добавим его в файл IReadWrite.inc (листинг 5.95).

Листинг 5.95. Содержимое файла C:\xampp\php\includes\main\IReadWrite.inc

```
<?php
namespace main;
interface IReadWrite extends IRead, IWrite {
}
?>
```

Теперь создадим класс `G`, который реализует этот интерфейс (листинг 5.96).

Листинг 5.96. Реализация интерфейса `IReadWrite`

```
class G implements IReadWrite {
   public function read(): string {
      return 'строка 4';
   }
   public function write(string $s): void {
      echo $s;
   }
}
```

Так как новый интерфейс содержит методы и для ввода, и для вывода, мы можем его использовать в нашем процессе ввода, обработки и вывода, как обычно:

```
$objG = new G();
$obj4 = new D($objG, $objG);
$obj4->change();
$obj4->write();                              // СТРОКА 4
```

Язык PHP допускает указание нескольких интерфейсов через запятую при реализации, а также множественное наследование интерфейсов. Если внутри двух интерфейсов окажутся два одноименных метода, но с разной сигнатурой, то будет выведено сообщение об ошибке. Однако если сигнатуры совпадают, то ошибки не будет.

5.13.5. Создание констант внутри интерфейса

Помимо сигнатур методов, в блоке интерфейса можно создавать общедоступные константы. Создадим интерфейс `IConst` и добавим его в файл IConst.inc (листинг 5.97).

Листинг 5.97. Содержимое файла C:\xampp\php\includes\main\IConst.inc

```
<?php
namespace main;
interface IConst {
   public const MY_CONST = 10;
   public function write(int $n) : void;
}
?>
```

В этом примере мы добавили в блок интерфейса константу и объявление метода. Класс, реализующий этот интерфейс (листинг 5.98), должен переопределить только метод `write()`. Константа просто наследуется классом и доступна через название класса или через название интерфейса:

```
echo IConst::MY_CONST; // 10
```

Листинг 5.98. Реализация интерфейса `IConst`

```
class H implements IConst {
   // Ошибка! Константу переопределить нельзя!
   // public const MY_CONST = 44;
   public function write(int $n): void {
      echo $n . "\n";
   }
}
```

Обратите внимание: переопределить константу внутри класса, реализующего интерфейс, нельзя. Попытка переопределить константу при наследовании интерфейса также приведет к ошибке.

Создадим экземпляр класса `H` и выведем значение константы:

```
$objH = new H();
$objH->write(H::MY_CONST);      // 10
$objH->write(IConst::MY_CONST); // 10
```

5.13.6. Интерфейсы и обратный вызов

Интерфейсы можно использовать также для создания обработчиков различных событий. В этом случае в функцию передается ссылка на объект, реализующий какой-либо интерфейс. Внутри блока интерфейса указывается сигнатура метода, который будет вызван при наступлении события. Нам достаточно в своем классе реализовать интерфейс и переопределить метод, указанный в интерфейсе. Далее создаем экземпляр класса и передаем его в метод регистрации обработчика события. При наступлении события остается лишь вызвать метод из интерфейса.

Пример реализации обратного вызова приведен в листинге 5.99.

Листинг 5.99. Интерфейсы и обратный вызов

```
<?php
declare(strict_types=1);
namespace main;
// Отключаем буферизацию
if (ob_get_level() > 0) ob_end_flush();
ob_implicit_flush();

interface IClick {
   public function onClick() : void;
}
class MyHandler implements IClick {
   public function onClick() : void {
      echo str_pad("Нажата кнопка<br>\n", 4096);
   }
}
```

```
class MyButton {
   private $ic = null;
   public function setIc(IClick $ic) { // Регистрация обработчика
      $this->ic = $ic;
   }
   public function click() : void {    // Нажатие кнопки
      if ( !is_null($this->ic) )
         $this->ic->onClick();
   }
}
$button = new MyButton();
$handler = new MyHandler();
$button->setIc($handler);             // Регистрация обработчика
for ($i = 0; $i < 5; $i++) {
   sleep(2);                          // Имитация ожидания
   $button->click();                  // Генерируем нажатие
}
```

В этом примере мы создали интерфейс `IClick` и говорим, что классы, реализующие интерфейс, должны содержать метод `onClick()`, который будет вызван при нажатии кнопки (имитация обработки событий в оконных приложениях) пользователем.

Далее создаем класс `MyHandler`, реализующий интерфейс, и переопределяем метод `onClick()`. Внутри этого метода просто выводим сообщение в окно Web-браузера таким образом, чтобы сообщение не попало в кэш. Внутри класса `MyButton` (описывающего кнопку) есть свойство, в котором хранится ссылка на обработчик события нажатия кнопки. С помощью метода `setIc()` производится регистрация обработчика, а метод `click()` — имитирует нажатие кнопки. Далее создаем экземпляр кнопки и экземпляр обработчика, а затем регистрируем обработчик и внутри цикла имитируем нажатие кнопки. Каждый раз в окно Web-браузера будет выводиться сообщение о факте нажатия кнопки.

5.13.7. Функции для работы с интерфейсами

Приведем функции, предназначенные для работы с интерфейсами:

- `interface_exists()` — возвращает значение `true`, если указанный интерфейс объявлен, и `false` — в противном случае. Название интерфейса указывается с предваряющим названием пространства имен. Формат функции:

  ```
  interface_exists(string $name[, bool $autoload=true]) : bool
  ```

  Пример:

  ```
  var_dump(interface_exists('main\\IClick'));  // bool(true)
  ```

- `get_declared_interfaces()` — возвращает список всех доступных сценарию интерфейсов (и пользовательских, и встроенных):

  ```
  print_r(get_declared_interfaces());
  ```

Проверить, реализует ли класс объекта указанный интерфейс, позволяет оператор `instanceof`:

```
$handler = new MyHandler();
if ($handler instanceof IClick) {
   echo 'Класс реализует интерфейс';
}
else echo 'Не реализует';
```

5.13.8. Сериализация объектов

Сериализация объектов выполняется с помощью следующих функций:

- `serialize()` — позволяет преобразовать объект в строку специального формата;
- `unserialize()` — используется для восстановления объекта из строки, преобразованной с помощью функции `serialize()`. Во втором параметре можно указать список с разрешенными названиями классов внутри ассоциативного массива с ключом `allowed_classes`. Если класса нет в списке, то создается объект класса `__PHP_Incomplete_Class`. Если во втором параметре указать значение `false`, то класс создаваться не будет, а если `true` или параметр не указан — то будут создаваться объекты любого класса. Формат функции:

  ```
  unserialize(string $str[, array $options]) : mixed
  ```

Пример сериализации объектов показан в листинге 5.100.

Листинг 5.100. Сериализация объектов

```
<?php
declare(strict_types=1);
namespace main;

class MyClass {
   public $n = 10;
   private $s = '';
   public function __construct($s) {
      $this->s = $s;
   }
   public function __toString() {
      return $this->n . ' ' . $this->s . "<br>\n";
   }
}
$obj = new MyClass('private data');
$str = serialize($obj);
echo $str . "<br>\n";
$obj2 = unserialize($str, ['allowed_classes' => ['main\\MyClass']]);
echo $obj2; // 10 private data
```

Следует учитывать, что при использовании функции `unserialize()` конструктор класса не вызывается.

Методы __sleep() и __wakeup()

Как видно из результата выполнения последней инструкции, сериализации подвергаются не только общедоступные свойства, но и закрытые. Чтобы этого избежать, нужно внутри класса создать магический метод `__sleep()`. Внутри метода нужно вернуть список с именами свойств, разрешенных для сериализации. Этот метод будет вызываться функцией `serialize()`.

Если внутри класса создать магический метод `__wakeup()`, то он будет вызван функцией `unserialize()` после создания объекта. Внутри этого метода можно воссоздать ресурсы объекта — например, выполнить подключение к базе данных.

Пример использования магических методов `__sleep()` и `__wakeup()` приведен в листинге 5.101.

Листинг 5.101. Методы `__sleep()` и `__wakeup()`

```
<?php
declare(strict_types=1);
namespace main;

class MyClass {
   public $n = 10;
   private $s = '';
   public function __construct($s) {
      $this->s = $s;
   }
   public function __sleep() {
      echo "__sleep<br>\n";
      return ['n'];
   }
   public function __wakeup() {
      echo "__wakeup()<br>\n";
   }
   public function __toString() {
      return $this->n . ' ' . $this->s . "<br>\n";
   }
}
$obj = new MyClass('private data');
$str = serialize($obj);
echo $str . "<br>\n";
$obj2 = unserialize($str, ['allowed_classes' => ['main\\MyClass']]);
echo $obj2; // 10
```

Интерфейс *Serializable*

Вместо добавления в класс методов `__sleep()` и `__wakeup()` можно реализовать интерфейс `Serializable`. В этом случае будут вызываться методы из интерфейса, а не методы `__sleep()` и `__wakeup()`. Однако если сериализация была ранее выполнена без использования методов из интерфейса, то будет вызван метод `__wakeup()`.

Интерфейс `Serializable` содержит объявление двух методов:

```
public function serialize();
public function unserialize($serialized);
```

- Метод `serialize()` вызывается функцией `serialize()`. Внутри метода нужно создать массив, содержащий значения свойств, подлежащих сериализации, и передать его функции `serialize()`. Метод должен вернуть результат выполнения сериализации массива.
- Метод `unserialize()` вызывается функцией `unserialize()`. Внутри метода нужно преобразовать переданную в качестве параметра строку специального формата в массив с помощью функции `unserialize()`. Затем следует выполнить инициализацию свойств класса значениями из этого массива. Иными словами, метод `unserialize()` выполняет роль конструктора.

Пример реализации интерфейса `Serializable` приведен в листинге 5.102.

Листинг 5.102. Пример реализации интерфейса `Serializable`

```
<?php
declare(strict_types=1);
namespace main;

class MyClass implements \Serializable {
   public $n = 10;
   private $s = '';
   public function __construct($s) {
      $this->s = $s;
   }
   public function serialize() {
      return serialize(['n' => $this->n, 's' => $this->s]);
   }
   public function unserialize($serialized) {
      $arr = unserialize($serialized);
      $this->n = $arr['n'] ?? 10;
      $this->s = $arr['s'] ?? '';
   }
   public function __toString() {
      return $this->n . ' ' . $this->s . "<br>\n";
   }
}
```

```
$obj = new MyClass('private data');
$str = serialize($obj);
echo $str . "<br>\n";
$obj2 = unserialize($str, ['allowed_classes' => ['main\\MyClass']]);
echo $obj2; // 10 private data
```

Обратите внимание: встроенные интерфейсы, так же как и классы, по умолчанию не видны внутри пользовательского пространства имен. Поэтому перед именем интерфейса нужно обязательно указать символ `\`.

5.13.9. Итераторы

Цикл `foreach` позволяет перебирать не только массивы, но и объекты, реализующие интерфейс `Traversable`. Этот интерфейс не содержит объявлений методов — он всего лишь является базовым для интерфейсов `IteratorAggregate` и `Iterator`.

Интерфейс *IteratorAggregate*

Интерфейс `IteratorAggregate` содержит объявление метода `getIterator()`:

```
interface IteratorAggregate extends Traversable {
   public function getIterator();
}
```

Внутри метода `getIterator()` нужно вернуть идентификатор внешнего итератора. В качестве примера вернем объект класса `ArrayIterator` (листинг 5.103).

Листинг 5.103. Пример реализации интерфейса `IteratorAggregate`

```
<?php
declare(strict_types=1);
namespace main;

class MyClass implements \IteratorAggregate {
   public $x = 10;
   public $y = 5;
   public $z = 22;
   private $s = 'private data';
   public function getIterator() {
      return new \ArrayIterator($this);
   }
}
$obj = new MyClass();
foreach ($obj as $key => $value) {
   echo "$key $value; ";
} // x 10; y 5; z 22;
```

Интерфейс *Iterator*

В интерфейсе `Iterator` объявлены следующие методы:

```
interface Iterator extends Traversable {
   public function next();
   public function valid();
   public function current();
   public function rewind();
   public function key();
}
```

Предназначение методов:

- `rewind()` — устанавливает указатель на первый элемент;
- `next()` — перемещает указатель на один элемент вперед;
- `key()` — возвращает ключ текущего элемента;
- `current()` — возвращает значение текущего элемента;
- `valid()` — возвращает значение `true`, если элемент существует, и `false` — в противном случае.

Пример реализации интерфейса `Iterator` приведен в листинге 5.104.

Листинг 5.104. Пример реализации интерфейса `Iterator`

```
<?php
declare(strict_types=1);
namespace main;

class MyClass implements \Iterator {
   private $arr = [10, 5, 8];
   private $pos = 0;
   public function next() {
      ++$this->pos;
   }
   public function valid() {
      return isset($this->arr[$this->pos]);
   }
   public function current() {
      return $this->arr[$this->pos];
   }
   public function rewind() {
      $this->pos = 0;
   }
   public function key() {
      return $this->pos;
   }
}
```

```
$obj = new MyClass();
foreach ($obj as $key => $value) {
   echo "$key $value; ";
} // 0 10; 1 5; 2 8;
```

5.14. Трейты

Как вы уже знаете, в объявлении класса после ключевого слова `extends` можно указать только один базовый класс, т. к. язык PHP не поддерживает множественное наследование. Однако можно имитировать множественное наследование — с помощью реализации нескольких интерфейсов, а также с помощью *трейтов*.

5.14.1. Создание и импорт трейта

Трейт описывается так же, как и класс, но вместо ключевого слова `class` используется ключевое слово `trait`:

```
trait <Имя трейта> {
   // Описание свойств и методов
}
```

Трейт очень похож на класс, но создать экземпляр трейта нельзя. Внутри фигурных скобок можно вставить описание обычных свойств и методов, а также описание статических членов и абстрактных методов. Обратите внимание: трейт не может содержать объявления констант.

Внутри обычных методов трейта доступен указатель `$this`. Доступ к статическим членам осуществляется с помощью ключевого слова `self`, а для вызова метода базового класса используется ключевое слово `parent`.

Выполнить импорт членов трейта в область видимости класса или другого трейта позволяет инструкция `use`. Форматы инструкции:

```
use <Имя трейта 1>[, ...[, <Имя трейта N>]];
use <Имя трейта 1>[, ...[, <Имя трейта N>]] {
   // Правила разрешения конфликтов
}
```

Пример использования трейтов приведен в листинге 5.105.

Листинг 5.105. Трейты

```
<?php
declare(strict_types=1);
namespace main;

trait MyTrait {
   public $x = 10;
   public static $static_property = 22;
```

```
   // public const MY_CONST = 3;  // Ошибка!
   public function test() {
      echo "MyTrait test(); x = {$this->x}<br>\n";
   }
   public static function static_func() {
      echo "MyTrait static_func(); static_property = ";
      echo self::$static_property . "<br>\n";
   }
   abstract public function func();
}
class MyClass {
   use MyTrait;                 // Импорт членов трейта
   public function func() {     // Переопределение абстрактного метода
      echo "func()<br>\n";
   }
}
$obj = new MyClass();
echo $obj->x;                   // Доступ к свойству
echo MyClass::$static_property; // Доступ к статическому свойству
$obj->test();                   // Вызов обычного метода
$obj->func();                   // Вызов обычного метода
echo MyClass::static_func();    // Вызов статического метода
```

При использовании статических свойств следует учитывать, что трейт и все классы, в которых выполнен импорт членов этого трейта, будут содержать независимые экземпляры этих свойств (листинг 5.106).

Листинг 5.106. Использование статических свойств

```
<?php
declare(strict_types=1);
namespace main;

trait MyTrait {
   public static $x = 2;
}
class MyClass {
   use MyTrait;
}
class MyClass2 {
   use MyTrait;
}
echo MyTrait::$x . ' ' . MyClass::$x . ' ' . MyClass2::$x . "<br>\n";
// 2 2 2
MyTrait::$x = 8;
echo MyTrait::$x . ' ' . MyClass::$x . ' ' . MyClass2::$x . "<br>\n";
```

```
// 8 2 2
MyClass::$x = 9;
echo MyTrait::$x . ' ' . MyClass::$x . ' ' . MyClass2::$x . "<br>\n";
// 8 9 2
MyClass2::$x = 1;
echo MyTrait::$x . ' ' . MyClass::$x . ' ' . MyClass2::$x . "<br>\n";
// 8 9 1
```

5.14.2. Импорт нескольких трейтов

В инструкции `use` можно указать сразу несколько трейтов через запятую. Если трейты содержат одноименные методы или свойства, то возникнет конфликт имен, и мы получим фатальную ошибку. Однако если значения свойств совпадают, то ошибки не будет.

Чтобы избежать фатальной ошибки, нужно в инструкции `use` после перечисления трейтов внутри фигурных скобок прописать правила разрешения конфликтов:

- `insteadof` — позволяет указать, из какого трейта нужно использовать идентификатор при конфликте имен:

```
// Использовать test из MyTrait вместо test из MyTrait2
MyTrait::test insteadof MyTrait2;
```

- `as` — позволяет создать псевдоним:

```
// Создать псевдоним test2 для test из MyTrait2
MyTrait2::test as test2;
```

Пример импорта нескольких трейтов с разрешением конфликта имен приведен в листинге 5.107.

Листинг 5.107. Импорт нескольких трейтов с разрешением конфликта имен

```
<?php
declare(strict_types=1);
namespace main;

trait MyTrait {
   public $x = 2;
   public function test() {
      echo __TRAIT__ . ' test() ';
   }
}
trait MyTrait2 {
   public $x = 2;  // Ошибки не будет, значения $x одинаковые
   public $y = 5;
   public function test() {
      echo __TRAIT__ . ' test() ';
   }
}
```

```
class MyClass {
   use MyTrait, MyTrait2 {
      // Использовать test из MyTrait вместо MyTrait2
      MyTrait::test insteadof MyTrait2;
      // Создать псевдоним test2 для test из MyTrait2
      MyTrait2::test as test2;
   }
}
$obj = new MyClass();
echo $obj->x;  // 2
echo $obj->y;  // 5
$obj->test();  // main\MyTrait test()
$obj->test2(); // main\MyTrait2 test()
```

5.14.3. Изменение модификатора доступа при импорте

Ключевое слово `as` в инструкции `use` позволяет не только создать псевдоним, но и изменить модификатор доступа. Пример изменения модификатора доступа приведен в листинге 5.108.

Листинг 5.108. Изменение модификатора доступа при импорте

```
<?php
declare(strict_types=1);
namespace main;

trait MyTrait {
   protected function test() {
      echo __TRAIT__ . ' test() ';
   }
   protected function func() {
      echo __TRAIT__ . ' func() ';
   }
}
class MyClass {
   use MyTrait {
      MyTrait::test as public;
      MyTrait::func as public test2;
   }
}
$obj = new MyClass();
$obj->test();  // main\MyTrait test()
$obj->test2(); // main\MyTrait func()
```

5.14.4. Приоритет при наследовании

При наследовании используются следующие правила разрешения конфликтов:

- если внутри трейта и базового класса существуют одноименные методы, имеющие одинаковую сигнатуру, то метод из трейта переопределит метод базового класса. Чтобы вызвать переопределенный метод базового класса, следует использовать ключевое слово `parent` (например, `parent::test()`);
- если внутри трейта и текущего класса существуют одноименные методы, имеющие одинаковую сигнатуру, то метод из текущего класса переопределит метод из трейта. Чтобы вызвать переопределенный метод из трейта, следует создать псевдоним в инструкции `use` (например, `MyTrait::func as func2`);
- если одноименные методы имеют разную сигнатуру, то это приведет к фатальной ошибке.

Пример учета приоритета при наследовании приведен в листинге 5.109.

Листинг 5.109. Приоритет при наследовании

```
<?php
declare(strict_types=1);
namespace main;

trait MyTrait {
   public function test() {
      echo __TRAIT__ . ' test() ';
      // Так можно вызвать одноименный метод из базового класса
      // parent::test();
   }
   public function func() {
      echo __TRAIT__ . ' func() ';
   }
}
class MyBaseClass {
   public function test() {
      echo __CLASS__ . ' test() ';
   }
}
class MyClass extends MyBaseClass {
   // Переопределяет test() из MyBaseClass
   use MyTrait {
      // Создание псевдонима для переопределенного метода
      MyTrait::func as func2;
   }
   // Переопределяет func() из MyTrait
   public function func() {
      echo __CLASS__ . ' func() ';
   }
}
```

```
$obj = new MyClass();
$obj->test();  // main\MyTrait test()
$obj->func();  // main\MyClass func()
$obj->func2(); // main\MyTrait func()
```

5.14.5. Импорт трейта внутри другого трейта

Инструкцию `use` допускается указывать не только внутри класса, но и внутри блока другого трейта. Рассмотрим это на примере (листинг 5.110).

Листинг 5.110. Импорт трейта внутри другого трейта

```
<?php
declare(strict_types=1);
namespace main;

trait MyTrait1 {
   public function test() {
      echo __TRAIT__ . ' test() ';
   }
}
trait MyTrait2 {
   public function func() {
      echo __TRAIT__ . ' func() ';
   }
}
trait MyTrait3 {
   use MyTrait1, MyTrait2;
}
class MyClass {
   use MyTrait3;
}
$obj = new MyClass();
$obj->test();  // main\MyTrait1 test()
$obj->func();  // main\MyTrait2 func()
```

5.14.6. Функции для работы с трейтами

Приведем функции, предназначенные для работы с трейтами:

- `trait_exists()` — возвращает значение `true`, если указанный трейт объявлен, и `false` — в противном случае. Название трейта указывается с предваряющим названием пространства имен. Формат функции:

  ```
  trait_exists(string $name[, bool $autoload=true]) : bool
  ```

  Пример:

  ```
  var_dump(trait_exists('main\\MyTrait1')); // bool(true)
  ```

- `get_declared_traits()` — возвращает список всех доступных сценарию трейтов:

  ```
  print_r(get_declared_traits());
  ```

5.15. Обработка ошибок

Существуют три типа ошибок в скриптах: синтаксические, логические и ошибки времени выполнения. Если ошибку не устранить или не обработать ее внутри скрипта, то интерпретатор либо выведет сообщение о фатальной ошибке и досрочно прервет выполнение программы, либо выведет предупреждающее сообщение, но при этом неправильный результат предыдущей инструкции может привести к следующей ошибке.

5.15.1. Синтаксические ошибки

Синтаксические ошибки — это ошибки в имени оператора или функции, отсутствие закрывающей или открывающей скобок и т. д., т. е. ошибки в синтаксисе языка. Как правило, интерпретатор предупредит о наличии ошибки, а программа не будет выполняться совсем.

Например, если вместо:

```
echo 'строка';
```

написать

```
ecgo 'строка';
```

то Web-браузер отобразит нечто подобное:

```
Parse error: syntax error, unexpected ''строка''
(T_CONSTANT_ENCAPSED_STRING) in C:\xampp\htdocs\index.php on line 22
```

Здесь интерпретатор предупреждает нас, что в строке `22` файла `index.php` содержится ошибка. Достаточно отсчитать 22 строки в исходном коде и исправить опечатку с `ecgo` на `echo`.

Приведем часто встречающиеся синтаксические ошибки:

- отсутствует точка с запятой в конце инструкции;
- опечатка в имени оператора или функции;
- буква набрана в русской раскладке клавиатуры вместо латинской;
- отсутствие открывающей или закрывающей скобки (или, наоборот, лишние скобки);
- в цикле `for` указаны параметры через запятую, а не через точку с запятой.

5.15.2. Логические ошибки

Логические ошибки — это ошибки в логике работы программы, которые можно выявить только по результатам ее выполнения. Как правило, интерпретатор не предупреждает о наличии логической ошибки, а программа будет выполняться, т. к.

не содержит синтаксических ошибок. Такие ошибки весьма трудно выявить, и их поиск часто заканчивается бессонными ночами.

Например, в логическом выражении вместо оператора == (равно) указан оператор присваивания =, или там, где нужно использовать оператор идентичности, используется оператор равенства. С точки зрения синтаксиса здесь все правильно, а вот с логикой могут быть проблемы:

```
var_dump(10 == '10 str'); // bool(true)
```

В этом примере строка преобразуется в число и сравниваться будут числа, а не строки. Результатом будет истина, хотя здравый смысл подсказывает обратное.

5.15.3. Ошибки времени выполнения

Ошибки времени выполнения возникают во время работы скрипта. Их причиной являются события, не предусмотренные программистом. В одних случаях ошибки времени выполнения являются следствием логических ошибок, а в других причиной могут стать внешние события, например: отсутствие прав для записи в файл, файл не найден (пользователь взял и удалил файл, посчитав, что он не нужен) и др. В этом случае интерпретатор генерирует либо предупреждающее сообщение, либо исключение. Если в коде не предусмотрена обработка исключения, то программа прерывается, и выводится сообщение об ошибке.

5.15.4. Оператор @

С помощью оператора `@` можно подавить вывод предупреждающих сообщений в выражении, которому он предшествует. Например, подавить вывод сообщения об ошибке деления на ноль:

```
$value = @(2 / 0);
```

или

```
@$value = 2 / 0;
var_dump($value); // float(INF)
```

Однако после этого значение `$value` не будет иметь смысла (в данном случае `$value` получит значение `INF`), т. е. сама ошибка устранена не будет.

Обратите внимание: с помощью оператора `@` можно подавить вывод сообщений только для ошибок уровней `Notice` (например, использование не определенной переменной) и `Warning` (например, деление на ноль). Подавить ошибки более высокого уровня оператор `@` не позволяет.

5.15.5. Управление отображением сообщений об ошибках

Задать степень обработки и протоколирования ошибок позволяет директива `error_reporting` в файле php.ini:

```
error_reporting=E_ALL
```

Приведем константы, указываемые в этой директиве:

```
var_dump(E_ALL);                // int(32767) (все ошибки)
var_dump(E_ERROR);              // int(1) (фатальные ошибки)
var_dump(E_WARNING);            // int(2) (предупреждения)
var_dump(E_PARSE);              // int(4) (синтаксические ошибки)
var_dump(E_NOTICE);             // int(8) (уведомления)
var_dump(E_DEPRECATED);         // int(8192) (устаревшие инструкции)
var_dump(E_STRICT);             // int(2048)
var_dump(E_CORE_ERROR);         // int(16)
var_dump(E_CORE_WARNING);       // int(32)
var_dump(E_COMPILE_ERROR);      // int(64)
var_dump(E_COMPILE_WARNING);    // int(128)
var_dump(E_RECOVERABLE_ERROR); // int(4096)
// Ошибки, генерируемые в скрипте:
var_dump(E_USER_ERROR);         // int(256)
var_dump(E_USER_WARNING);       // int(512)
var_dump(E_USER_NOTICE);        // int(1024)
var_dump(E_USER_DEPRECATED);    // int(16384)
```

Чаще всего указывается комбинация этих констант через побитовые операторы:

```
error_reporting=E_ALL & ~E_NOTICE
```

Знак `~` (тильда), стоящий перед значением, указывает, что вывод сообщений об ошибке этого типа должен быть выключен. Иными словами, прочитать указанное значение можно так: включить вывод всех сообщений об ошибках, кроме уведомлений.

Если доступа к файлу php.ini нет (на виртуальном хостинге доступа точно не будет), то в сценарии можно вызвать функцию `error_reporting()`:

```
error_reporting(E_ALL & ~E_NOTICE & ~E_WARNING);
```

В качестве параметра функции `error_reporting()` можно указать число:

```
error_reporting(32757);
var_dump(E_ALL & ~E_NOTICE & ~E_WARNING); // int(32757)
```

Чтобы отключить вывод сообщений, достаточно указать значение `0`:

```
error_reporting(0);
```

Включить вывод всех сообщений об ошибках можно с помощью константы `E_ALL` или значения `-1`:

```
error_reporting(-1);
```

Если параметр не задан, то функция вернет текущее значение:

```
echo error_reporting(); // 32767
var_dump(E_ALL);        // int(32767)
```

Изменить значение директивы `error_reporting` из скрипта можно также с помощью функции `ini_set()`:

```
ini_set('error_reporting', E_ALL & ~E_NOTICE);
echo error_reporting();          // 32759
var_dump(E_ALL & ~E_NOTICE);     // int(32759)
```

Для изменения значения директивы `error_reporting` из файла .htaccess используется следующий синтаксис:

```
php_value error_reporting 32759
```

На виртуальном хостинге принято не выводить ошибки в Web-браузер, а записывать их в журнал ошибок error.log. В этом случае при возникновении фатальной ошибки пользователь увидит белый экран, а не сообщение об ошибке, в составе которой может содержаться конфиденциальная информация.

Отключить вывод ошибок в Web-браузер позволяет директива `display_errors` в файле php.ini:

```
display_errors=Off
```

Директива `log_errors` включает вывод сообщений об ошибках в журнал ошибок:

```
log_errors=On
```

Задать путь к файлу, в который будут выводиться ошибки, позволяет директива `error_log`:

```
error_log="C:/xampp/php/logs/php_error_log"
```

Изменить эти директивы из скрипта можно с помощью функции `ini_set()`:

```
ini_set('error_reporting', E_ALL);
ini_set('display_errors', 'Off');
ini_set('error_log', 'php_error_log');
ini_set('log_errors', 'On');
```

Для изменения значения директив из файла .htaccess используется следующий синтаксис:

```
php_value error_reporting 32767
php_value display_errors "0"
php_flag log_errors On
php_value error_log "php_error_log"
```

Обратите внимание: мы не указали путь в последней инструкции, поэтому файл будет создаваться в том же каталоге, что и файл со скриптом. Если нужно все сообщения записывать в один файл, то следует указать путь.

В предыдущих версиях PHP директива `display_errors` имела логическое значение, а не строку, как в PHP 7, поэтому использовалась следующая инструкция для отключения вывода сообщений об ошибках в Web-браузер:

```
php_flag display_errors Off
```

5.15.6. Инструкция *or die()*

Для обработки критических для всей программы ошибок можно использовать инструкцию `or die()`. В круглых скобках может быть указано сообщение об ошибке или функция, которая будет вызвана при возникновении ошибки. После вывода сообщения или вызова функции выполнение скрипта прекратится:

```
@$file = fopen("file.txt", "r") or die("Ошибка");
```

или

```
@$file = fopen("file.txt", "r") or die(err_msg());
function err_msg() {
   echo "Ошибка";
}
```

5.15.7. Обработка и генерация пользовательских ошибок

Функция `set_error_handler()` позволяет назначить пользовательский обработчик ошибок. Формат функции:

```
set_error_handler(callable $error_handler[, int $error_types]) : mixed
```

В первом параметре указывается функция обратного вызова, которая будет использоваться для обработки ошибок. Формат функции:

```
function ($errno, $errstr, $errfile, $errline)
```

Через параметр `$errno` доступен уровень ошибки, через `$errstr` — описание ошибки в виде строки, через `$errfile` — название файла, а через `$errline` — номер строки с ошибкой. Функция должна возвращать значение `true`, если обработка успешно выполнена, и `false` — в противном случае. В первом случае управление будет передано следующей инструкции после инструкции, вызвавшей ошибку, а во втором — стандартному обработчику ошибок.

Обратите внимание: во-первых, не все типы ошибок можно обработать, во-вторых, если продолжение работы лишено смысла, то внутри обработчика нужно вызвать оператор `exit()` или передать управление стандартному обработчику ошибок, вернув значение `false`.

Функция `trigger_error()` позволяет сгенерировать пользовательскую ошибку внутри программы. Формат функции:

```
trigger_error(string $error_msg[, int $error_type]) : bool
```

В первом параметре указывается текст описания ошибки длиной до 1024 байтов. Во втором параметре можно указать тип ошибки. По умолчанию используется тип `E_USER_NOTICE`.

Обработаем ошибки `E_NOTICE` и `E_USER_NOTICE`, а остальные ошибки передадим стандартному обработчику ошибок (листинг 5.111).

Листинг 5.111. Обработка и генерация ошибок

```
<?php
set_error_handler(function ($errno, $errstr, $errfile, $errline) {
   switch ($errno) {
      case E_NOTICE:
         echo "Ошибка E_NOTICE; текст: $errstr; файл: $errfile; ";
         echo "строка: $errline<br>\n";
         // Стандартный обработчик вызван не будет
         return true;
      case E_USER_NOTICE:
         echo "Ошибка E_USER_NOTICE; текст: $errstr<br>\n";
         // Стандартный обработчик вызван не будет
         return true;
   }
   // В других случаях передаем управление стандартному обработчику
   return false;
});
echo $x;       // Ошибка E_NOTICE      (обрабатываем)
$x = 10 / 0;   // Ошибка E_WARNING     (не обрабатываем)
               // Ошибка E_USER_NOTICE (обрабатываем)
trigger_error("Описание ошибки", E_USER_NOTICE);
```

Результат в окне Web-браузера:

```
Ошибка E_NOTICE; текст: Undefined variable: x; файл:
C:\xampp\htdocs\index.php; строка: 17

Warning: Division by zero in C:\xampp\htdocs\index.php on line 18
Ошибка E_USER_NOTICE; текст: Описание ошибки
```

5.15.8. Инструкция *try...catch...finally*

Помимо обработки ошибок, язык PHP содержит возможность обработки и генерации исключений. *Исключения* — это извещения, возбуждаемые в случае возникновения ошибки в программном коде или при наступлении какого-либо события. Если в коде не предусмотрена обработка исключения, то выполнение программы прерывается, и выводится сообщение об ошибке.

Для обработки исключений предназначена инструкция `try...catch`. Формат инструкции:

```
try {
    <Блок, в котором перехватывается исключение>
}
[catch (<Класс исключения1> <Переменная>) {
    <Блок, выполняемый при возникновении исключения>
}
```

```
[...
catch (<Класс исключенияN> <Переменная>) {
    <Блок, выполняемый при возникновении исключения>
}]]
[finally {
    <Блок, выполняемый в любом случае>
}]
```

Инструкции, в которых перехватываются исключения, должны быть расположены внутри блока `try`. Если при выполнении этих инструкций возникнет исключение, то управление будет передано в блок `catch`, который соответствует классу исключения. Классом исключения может выступать встроенный класс или пользовательский класс. Обратите внимание: если исключение не возникло, то инструкции внутри блока `catch` не выполняются.

Начиная с PHP 7.1, вместо нескольких блоков `catch` можно использовать один блок `catch` с несколькими исключениями, указанными через символ `|`. В этом случае управление будет передано в блок `catch` при генерации любого из этих исключений:

```
try {
   //...
} catch (\Error | \Exception $e) {
   echo "Error или Exception";
}
```

В качестве примера использования инструкции `try...catch` инсценируем создание объекта не существующего класса и обработаем исключение:

```
<?php
declare(strict_types=1);
namespace main;
try {
   $obj = new MyClass();
   echo 'Инструкция выполнена не будет!';

} catch (\Error $e) {
   echo "Error: не удалось создать объект<br>\n";
}
echo 'Инструкция после try';
```

Результат в окне Web-браузера:

```
Error: не удалось создать объект
Инструкция после try
```

При возникновении исключения внутри блока `try` управление сразу передается в блок `catch`, соответствующий классу возникшего исключения. В нашем примере классом исключения является встроенный класс `Error`. Код, расположенный после инструкции, сгенерировавшей исключение, выполнен не будет, поэтому вывод сообщения мы не увидим, — после выполнения инструкций в блоке `catch` управление

передается инструкции, расположенной сразу после инструкции `try...catch`. Иными словами, считается, что исключение обработано, и можно продолжить выполнение программы.

В некоторых случаях — например, если не имеет смысла продолжать работу, — необходимо не продолжить выполнение программы, а прервать ее выполнение после перехвата исключения. В этом случае можно внутри блока `catch` произвести завершающие действия (например, проинформировать пользователя о проблеме), а затем прервать работу программы с помощью оператора `exit()`:

```
<?php
declare(strict_types=1);
namespace main;
try {
   $obj = new MyClass();
   echo 'Инструкция выполнена не будет!';

} catch (\Error $e) {
   exit("Error: не удалось создать объект");
}
echo 'Инструкция после try';
```

Результат в окне Web-браузера:

```
Error: не удалось создать объект
```

Если нет блока `catch`, соответствующего классу исключения, то исключение «всплывает» к обработчику более высокого уровня. Если исключение в программе нигде не обрабатывается, то управление передается обработчику по умолчанию, который останавливает выполнение программы и выводит стандартную информацию об ошибке. Таким образом, в обработчике может быть несколько блоков `catch` с разными классами исключений. Кроме того, один обработчик можно вложить в другой (листинг 5.112).

Листинг 5.112. Обработка исключений

```
<?php
declare(strict_types=1);
namespace main;
class MyException extends \Exception {}
try {
   try {
      $obj = new MyClass();
      // throw new MyException();
   } catch (MyException $ex) {
      echo "MyException<br>\n";
   } catch (\Exception $ex) {
      echo "Exception: {$ex->getMessage()}<br>\n";
   }
```

```
    echo 'Инструкция после вложенного обработчика';
} catch (\Error $e) { // Выполняется этот блок!
    echo "Error: не удалось создать объект<br>\n";
}
echo 'Инструкция после try';
```

В этом примере во вложенном обработчике не указано исключение `Error`, поэтому исключение «всплывает» к обработчику более высокого уровня. После обработки исключения управление передается инструкции, расположенной сразу после обработчика. В нашем примере управление будет передано инструкции, выводящей сообщение `Инструкция после try`. Обратите внимание на то, что инструкция:

```
echo 'Инструкция после вложенного обработчика';
```

выполнена не будет.

При указании нескольких блоков `catch` с разными классами исключений следует учитывать иерархию классов исключений. Если в одной инструкции `try...catch` находятся блоки `catch` с базовым и производным классами, то вначале должен идти блок с производным классом, а затем блок с базовым, иначе блок с производным классом никогда не будет выполнен. В нашем случае вначале идет блок с классом `MyException`, а лишь затем блок с классом `Exception`.

Если в блоке `catch` указан самый верхний класс в иерархии классов исключений, то он будет перехватывать исключения любых производных классов. Если в блоке `catch` указан интерфейс `Throwable`, который реализуют и класс `Exception`, и класс `Error`, то можно перехватить все исключения (листинг 5.113).

Листинг 5.113. Перехват всех исключений

```
<?php
declare(strict_types=1);
namespace main;
try {
    $obj = new MyClass();
} catch (\Throwable $e) {
    echo "Обработка любых исключений в коде<br>\n";
}
echo 'Инструкция после try';
```

Обратите внимание: можно перехватить все исключения, но не все ошибки. Между этими двумя понятиями в PHP есть разница. Большинство встроенных функций генерируют именно ошибки и предупреждения, а не исключения. Поэтому перехватить ошибку деления на ноль с помощью инструкции `try...catch` нельзя.

Получить информацию об обрабатываемом исключении можно через переменную, объявленную в блоке `catch` (листинг 5.114).

Листинг 5.114. Получение информации об исключении

```
<?php
declare(strict_types=1);
namespace main;
try {
   $obj = new MyClass();
} catch (\Error $e) {
   echo "Error: не удалось создать объект<br>\n";
   echo $e->getMessage() . "<br>\n";
   echo $e->getFile() . "<br>\n";
   echo $e->getLine() . "<br>\n";
   print_r($e);
}
```

Результат в окне Web-браузера:

```
Error: не удалось создать объект
Class 'main\MyClass' not found
C:\xampp\htdocs\index.php
5
Error Object ( [message:protected] => Class 'main\MyClass' not found
[string:Error:private] => [code:protected] => 0 [file:protected] =>
C:\xampp\htdocs\index.php [line:protected] => 5
[trace:Error:private] => Array ( ) [previous:Error:private] => )
```

Помимо блоков `catch` инструкция `try...catch` может содержать блок `finally`, инструкции внутри которого выполняются вне зависимости от того, возникло в блоке `try` исключение или нет. Обычно внутри этого блока выполняется освобождение каких-либо ресурсов, например закрытие файлов. Если внутри блока `catch` вызвать оператор `exit()`, то инструкции внутри блока `finally` выполнены не будут. Пример использования блока `finally` приведен в листинге 5.115.

Листинг 5.115. Блок `finally`

```
<?php
declare(strict_types=1);
namespace main;
try {
   $obj = new MyClass();
} catch (\Error $e) {
   echo "Error: не удалось создать объект<br>\n";
} finally {
   echo "Блок finally<br>\n";
}
```

Результат в окне Web-браузера:

```
Error: не удалось создать объект
Блок finally
```

Если существует блок `finally`, то блоков `catch` может не быть вовсе:

```
<?php
declare(strict_types=1);
namespace main;
try {
   $obj = new MyClass();
} finally {
   echo "Блок finally<br>\n";
}
```

Результат в окне Web-браузера:

```
Блок finally

Fatal error: Uncaught Error: Class 'main\MyClass' not found in
C:\xampp\htdocs\index.php:5 Stack trace: #0 {main} thrown in
C:\xampp\htdocs\index.php on line 5
```

5.15.9. Оператор *throw*. Генерация исключений

Вместо обработки исключения можно повторно сгенерировать исключение внутри блока `catch`, чтобы передать управление вышестоящему обработчику. Например, можно сгенерировать исключение более высокого уровня. Кроме того, если мы разрабатываем какой-либо метод и внутри него не имеем возможности вернуть корректное значение при некорректном значении параметров, то лучше сгенерировать исключение, чтобы другие пользователи знали о проблеме. Например, метод возвращает целое число. Какое значение мы могли бы вернуть при ошибке? `-1`? Но, это также число. Генерация исключения в этом случае — лучшее решение.

Для генерации исключения в программе предназначен оператор `throw`. После оператора указывается объект класса исключения. Это может быть встроенный класс исключений или пользовательский класс, наследующий один из классов встроенных исключений. Давайте создадим два обработчика, один из которых вложен в другой. Внутри блока `try` вложенного обработчика имитируем создание объекта не существующего класса. Внутри блока `catch` вложенного обработчика перехватим это исключение и сгенерируем исключение другого класса. Повторное исключение перехватим в блоке `catch` внешнего обработчика (листинг 5.116).

Листинг 5.116. Оператор `throw`

```
<?php
declare(strict_types=1);
namespace main;
try {
   try {
      $obj = new MyClass();
   } catch (\Error $e) {
```

```
        throw new \Exception('Не удалось создать объект');
    }
    echo 'Инструкция после вложенного обработчика';
} catch (\Exception $ex) {
    echo "Exception: {$ex->getMessage()}<br>\n";
}
```

Результат в окне Web-браузера:

```
Exception: Не удалось создать объект
```

Обратите внимание на инструкцию:

```
throw new \Exception('Не удалось создать объект');
```

Здесь мы создаем экземпляр класса `Exception` и передаем конструктору класса строку с описанием ошибки. Эту строку мы можем получить внутри блока `catch` внешнего обработчика через переменную `$ex` с помощью метода `getMessage()`.

5.15.10. Иерархия классов исключений

Базовыми классами всех исключений являются классы `Error` и `Exception`. Класс `Error` описывает внутренние ошибки PHP, а класс `Exception` — ошибки внутри программы. Если мы в блоке `catch` указываем имя базового класса, то блок будет перехватывать исключения всех производных классов.

Формат конструкторов классов `Error` и `Exception`:

```
public __construct([$message=""[, $code=0[, $previous=null]]])
```

В параметре `$message` можно указать текст сообщения, который будет доступен через метод `getMessage()`. Параметр `$code` позволяет указать код исключения, который будет доступен через метод `getCode()`. Параметр `$previous` используется для создания цепочки исключений.

Вот пример генерации и обработки исключения класса `Exception`:

```
try {
    throw new \Exception('Сообщение об ошибке');
} catch (\Exception $ex) {
    echo "Exception: {$ex->getMessage()}<br>\n";
}
```

Все базовые классы исключений реализуют интерфейс `Throwable`. Если в блоке `catch` указать название этого интерфейса, то можно перехватить все исключения.

Интерфейс `Throwable` содержит следующие методы:

- `__toString()` — возвращает информацию об исключении в виде строки:

  ```
  $ex = new \Exception('Сообщение');
  echo $ex;
  // Exception: Сообщение in C:\xampp\htdocs\index.php:4
  // Stack trace: #0 {main}
  ```

- `getMessage()` — возвращает описание ошибки:

```
$ex = new \Exception('Сообщение об ошибке');
echo $ex->getMessage(); // Сообщение об ошибке
```

- `getCode()` — возвращает код исключения:

```
$ex = new \Exception('Сообщение об ошибке', 2);
echo $ex->getCode(); // 2
```

- `getFile()` — возвращает название файла:

```
$ex = new \Exception('Сообщение об ошибке');
echo $ex->getFile(); // C:\xampp\htdocs\index.php
```

- `getLine()` — возвращает номер строки:

```
$ex = new \Exception('Сообщение об ошибке');
echo $ex->getLine(); // 4
```

- `getPrevious()` — возвращает объект предыдущего исключения или `null`:

```
$e = new \Exception('Сообщение');
$ex = new \Exception('', 0, $e);
echo $ex->getPrevious()->getMessage(); // Сообщение
```

- `getTraceAsString()` — возвращает стек вызовов в виде строки;
- `getTrace()` — возвращает массив со стеком вызовов:

```
function test() {
   throw new \Exception();
}
try {
   test();
} catch (\Exception $ex) {
   echo $ex->getTraceAsString();
   // #0 C:\xampp\htdocs\index.php(8): main\test()
   print_r($ex->getTrace());
   // #1 {main}
   // Array ([0] => Array ([file] => C:\xampp\htdocs\index.php
   // [line] => 8 [function] => main\test [args] => Array ()))
}
```

5.15.11. Пользовательские классы исключений

Как вы уже знаете, классом исключения может выступать встроенный класс исключения или пользовательский класс, наследующий встроенный класс исключения. Основное преимущество использования классов для обработки исключений заключается в возможности указания базового класса для перехвата всех исключений соответствующих классов-потомков. Обратите внимание: блок `catch`, в котором указан объект производного класса, должен быть расположен перед блоком `catch`, в котором указан объект базового класса.

При разработке пользовательских классов исключений обычно наследуется класс `Exception` (листинг 5.117).

Листинг 5.117. Пользовательские классы исключений

```
<?php
declare(strict_types=1);
namespace main;
class MyException extends \Exception {
   public function __construct($msg="", $code=0, $prev=null) {
      if (strlen($msg) === 0) {
         $msg = 'Ошибка класса MyException';
      }
      parent::__construct($msg, $code, $prev);
   }
}
try {
   throw new MyException();
} catch (MyException $ex) {
   echo $ex->getMessage();
}
```

Результат выполнения:

```
Ошибка класса MyException
```

5.15.12. Способы поиска ошибок в программе

В предыдущих разделах мы научились обрабатывать ошибки времени выполнения. Однако наибольшее количество времени программист затрачивает на другой тип ошибок — логические ошибки. В этом случае программа запускается без ошибок, но результат выполнения программы не соответствует ожидаемому результату. Ситуация еще более осложняется, когда неверный результат проявляется лишь периодически, а не постоянно. Инсценировать ту же ситуацию, чтобы получить этот же неверный результат, бывает крайне сложно, и это занимает очень много времени. В этом разделе мы рассмотрим лишь основные способы поиска ошибок, а также дадим несколько советов по оформлению кода, что будет способствовать быстрому поиску ошибок.

Первое, на что следует обратить внимание, — это форматирование кода. Начинающие программисты обычно не обращают на это никакого внимания, считая этот процесс лишним. А на самом деле зря! Интерпретатору абсолютно все равно, разместите вы все инструкции на одной строке или выполните форматирование кода. Однако при поиске ошибок форматирование кода позволит найти ошибку гораздо быстрее.

Перед всеми инструкциями внутри блока должно быть расположено одинаковое количество пробелов. Обычно ставятся три или четыре пробела. От применения

символов табуляции лучше отказаться. Если все же вы их используете, то не следует в одном файле совмещать и пробелы, и табуляцию. Для вложенных блоков количество пробелов умножают на уровень вложенности: если для блока первого уровня вложенности использовались три пробела, то для блока второго уровня вложенности должно использоваться шесть пробелов, для третьего уровня — девять и т. д. Пример форматирования вложенных блоков приведен в листинге 5.118.

Листинг 5.118. Пример форматирования вложенных блоков

```
$arr =
[
   [1, 2, 3, 4],
   [5, 6, 7, 8]
];
foreach ($arr as $a) {
   foreach ($a as $value) {
      echo "$value ";
   }
   echo "<br>\n";
}
```

Открывающая фигурная скобка может быть расположена как на одной строке с оператором, так и на следующей строке. Какой способ использовать, зависит от предпочтений программиста или от требований по оформлению кода, принятых внутри фирмы.

Вот пример размещения открывающей фигурной скобки на отдельной строке:

```
foreach ($arr as $a)
{
   foreach ($a as $value)
   {
      echo "$value ";
   }
   echo "<br>\n";
}
```

Длина одной строки не должна содержать более 80 символов. Если количество символов больше, то следует выполнить переход на новую строку. При этом продолжение смещается относительно основной инструкции на величину отступа или выравнивается по какому-либо элементу. Иначе приходится пользоваться горизонтальной полосой прокрутки, а это очень неудобно при поиске ошибок.

Если программа слишком большая, то следует задуматься о разделении программы на отдельные функции или классы, которые выполняют логически законченные действия. Помните, что отлаживать отдельную функцию гораздо легче, чем «спагетти»-код. Причем, прежде чем вставить функцию (или класс) в основную программу, ее следует протестировать в отдельном проекте, передавая функции различные значения и проверяя результат выполнения.

Обратите внимание на то, что форматирование кода должно выполняться при его написании, а не во время поиска ошибок. Этим вы сократите время поиска ошибки и, скорее всего, заметите ошибку еще на этапе создания кода. Если все же ошибка возникла, то вначале следует инсценировать ситуацию, при которой ошибка проявляется. После этого можно начать поиск ошибки.

Причиной периодических ошибок чаще всего являются внешние данные. Например, если числа получаются от пользователя, а затем производится деление чисел, то вполне возможна ситуация, при которой пользователь введет число `0`. Деление на ноль приведет к неправильному результату и предупреждающему сообщению. Следовательно, все данные, которые поступают от пользователей, должны проверяться на соответствие допустимым значениям. Если данные не соответствуют, то нужно вывести сообщение об ошибке, а затем повторно запросить новое число или прервать выполнение всей программы.

Функции `print_r()` и `var_dump()` удобно использовать для вывода промежуточных значений переменных. В этом случае значения переменных вначале выводятся в самом начале программы и производится проверка соответствия значений. Если значения соответствуют, то инструкция с выводом перемещается на следующую строку программы и опять производится проверка и т. д. Если значения не совпали, то ошибка возникает в инструкции, расположенной перед инструкцией с выводом значения. Если ошибка допущена в пользовательской функции, то проверку значений производят внутри функции, каждый раз перемещая инструкцию с выводом значений. На одном из этих многочисленных этапов ошибка обычно обнаруживается. В больших программах можно интуитивно догадаться о примерном расположении инструкции с ошибкой и начать поиск ошибки оттуда, а не с самого начала программы.

Не забывайте, что интерпретатор всячески пытается помочь вам избежать ошибок в коде. Обязательно на этапе написания программы или при отладке включите максимальный режим отображения различных предупреждающих сообщений. Для этого достаточно добавить в самое начало программы следующие инструкции:

```
<?php
declare(strict_types=1);
error_reporting(E_ALL);
ini_set('display_errors', 'On');
```

На этапе работы приложения следует отключить вывод ошибок в Web-браузер, чтобы не показывать конфиденциальную информацию пользователю, но при этом включить запись ошибок в файл:

```
<?php
ini_set('error_reporting', E_ALL);
ini_set('display_errors', 'Off');
ini_set('error_log', 'php_error_log');
ini_set('log_errors', 'On');
```

Просматривайте этот файл регулярно. Как только в нем появились записи или какие-либо предупреждающие сообщения отобразились в окне Web-браузера, сразу

выполняйте поиск ошибок. В тексте сообщения указывается название файла и номер строки, в которой содержится инструкция с ошибкой:

```
[15-Oct-2017 05:25:27 Europe/Moscow] PHP Warning: Division by zero in
C:\xampp\htdocs\index.php on line 9
```

Как только вы увидели такую запись в файле, то вначале прочитайте описание ошибки. В приведенном примере говорится, что выполнено деление на ноль. Откройте файл C:\xampp\htdocs\index.php, отсчитайте 9 строк и посмотрите, что не так. Редактор Notepad++ позволяет пронумеровать инструкции, поэтому достаточно перейти к указанной в сообщении строке. Измените инструкцию таким образом, чтобы выполнить деление на ноль было невозможно.

В некоторых случаях в строке, на которую указывает интерпретатор, нет ошибок. Например, в этом коде интерпретатор указывает на строку 5, хотя ошибка содержится в строке 4 (нет точки с запятой после инструкции):

```
<?php
$arr = [1, 2, 3];
foreach ($arr as $value) {
   echo $value
}
```

Многие специализированные редакторы содержат специальное приложение, называемое *отладчиком*, с помощью которого можно сделать процесс поиска ошибок очень эффективным. Например, отладчик позволяет выполнять программу по шагам, при этом контролируя значения переменных на каждом шаге. Отладчик также позволяет проверить, соответствует ли порядок выполнения инструкций разработанному ранее алгоритму, что очень полезно при поиске логических ошибок.

Применение отладки — это самый эффективный способ нахождения ошибок, не требующий вставки никаких инструкций вывода промежуточных значений в текст программы. В процессе отладки мы и так на каждом шаге можем наблюдать за значениями всех или только избранных переменных. А после завершения отладки вам не придется удалять или временно отключать какие-либо инструкции. Пользуйтесь отладкой при возникновении любой ошибки, и вы очень быстро ее найдете и исправите.

5.16. Работа с файлами и каталогами

Очень часто нужно сохранить какие-либо данные. Для этого существуют два способа: сохранение в файл и сохранение в базе данных. Первый способ подходит для сохранения информации небольшого объема. Если объем велик, то лучше (и удобнее) воспользоваться базой данных.

Файлы используются при создании гостевых книг, списков рассылки, ленты новостей, протоколирования различных ситуаций (например, ошибок) и во многих других случаях.

Для чтения или записи файла нужно выполнить следующие действия:

1. Открыть файл.
2. Блокировать файл.
3. Считать или записать данные.
4. Сбросить буфер в файл (если что-то записывали).
5. Снять блокировку.
6. Закрыть файл.

5.16.1. Открытие и закрытие файла

Для открытия и закрытия файла предназначены следующие функции:

- fopen(<Путь к файлу>, <Режим>[, <Путь поиска>[, <Контекст>]]) — открывает файл и возвращает дескриптор (идентификатор). Параметр <Режим> может принимать следующие значения:
  - r — только чтение. После открытия файла указатель устанавливается на начало файла. Если файл не существует, функция fopen() вернет false;
  - r+ — чтение и запись. После открытия файла указатель устанавливается на начало файла. Если файл не существует, функция fopen() вернет false;
  - w — запись. Если файл не существует, то он будет создан. Если файл существует, то он будет перезаписан. После открытия файла указатель устанавливается на начало файла;
  - w+ — чтение и запись. Если файл не существует, то он будет создан. Если файл существует, то он будет перезаписан. После открытия файла указатель устанавливается на начало файла;
  - a — запись. Если файл не существует, то он будет создан. После открытия файла указатель устанавливается на конец файла. Содержимое файла не удаляется;
  - a+ — чтение и запись. После открытия файла указатель устанавливается на конец файла. Если файл не существует, то он будет создан. Содержимое файла не удаляется;
  - x — запись. Если файл не существует, то он будет создан. После открытия файла указатель устанавливается на начало файла. Если файл существует, то функция fopen() вернет false;
  - x+ — чтение и запись. Если файл не существует, то он будет создан. После открытия файла указатель устанавливается на начало файла. Если файл существует, то функция fopen() вернет false;
  - c — запись. Если файл не существует, то он будет создан. После открытия файла указатель устанавливается на начало файла. Содержимое файла не удаляется;

- c+ — чтение и запись. После открытия файла указатель устанавливается на начало файла. Если файл не существует, то он будет создан. Содержимое файла не удаляется.

Обратите внимание: при возврате функцией fopen() значения false дополнительно генерируется предупреждение уровня E_WARNING. Подавить это предупреждение позволяет оператор @. Если файл успешно открыт, то функция возвращает дескриптор ресурса:

```
@$file = fopen("test_file.txt", "r");
var_dump($file); // bool(false)
@$file = fopen("test_file.txt", "w");
var_dump($file); // resource(3) of type (stream)
var_dump(is_resource($file)); // bool(true)
```

Кроме того, после указания режима может следовать модификатор:

- b — файл будет открыт в бинарном режиме;
- t — файл будет открыт в текстовом режиме.

- fclose(<Дескриптор>) — закрывает файл.

5.16.2. Установка и снятие блокировки

Функция flock(<Дескриптор>, <Режим>[, &$wouldblock]) позволяет блокировать файл или снять блокировку. Если блокировка успешно установлена, функция вернет значение true. Параметр <Режим> может принимать следующие значения:

- LOCK_SH или 1 — разделяемый доступ для чтения. Если другой процесс хочет записать что-либо в файл, то ему придется подождать снятия блокировки;
- LOCK_EX или 2 — монопольный режим для записи. Файл не доступен для совместного использования;
- LOCK_UN или 3 — снимает блокировку.

Если файл блокирован, то процесс будет ждать снятия блокировки. Чтобы изменить это поведение, следует дополнительно указать константу LOCK_NB:

```
flock($file, LOCK_EX | LOCK_NB);
```

5.16.3. Чтение и запись файлов

Для записи данных в файл предназначены следующие функции:

- fwrite(<Дескриптор>, <Строка>[, <Длина>]) — записывает данные в файл и возвращает число записанных байтов или false — в случае ошибки.
- fflush(<Дескриптор>) — сбрасывает содержимое буфера в файл. Вот пример записи в файл:

```
@$file = fopen('test_file.txt', 'wb') or die('Ошибка');
if (flock($file, LOCK_EX)) { // Устанавливаем блокировку
   fwrite($file, 'Строка');  // Записываем данные
```

```
   fflush($file);              // Сбрасываем буфер
   flock($file, LOCK_UN);      // Снимаем блокировку
}
else echo 'Не удалось установить блокировку';
fclose($file);                 // Закрываем файл
```

- `fprintf(<Дескриптор>, <Формат>[, <Данные>])` — записывает данные в файл в соответствии со строкой формата. Спецификаторы, используемые в строке формата, мы рассматривали в *разд. 5.7.18*:

```
@$file = fopen('filetmp.txt', 'wb');
fprintf($file, '%.2f', 10.5125484);
fclose($file);
```

- `file_put_contents()` — записывает данные в файл. Если файл не существует, то он будет создан. Если файл существует, то по умолчанию он будет перезаписан. Для записи в конец файла следует указать флаг `FILE_APPEND`. Вызов функции эквивалентен последовательности вызовов функций `fopen()`, `fwrite()` и `fclose()`. Формат функции:

```
file_put_contents(<Путь к файлу>, <Данные>[, <Флаг>[, <Контекст>]])
```

  В параметре `<Флаг>` могут быть указаны следующие значения (или их комбинация):

  - `FILE_USE_INCLUDE_PATH` — поиск файла будет осуществлен в каталогах, перечисленных в директиве `include_path`;
  - `FILE_APPEND` — если файл существует, то данные будут добавлены в конец содержимого файла;
  - `LOCK_EX` — монопольный режим для записи. Файл не доступен для совместного использования.

  Пример:

```
// Перезапись файла
file_put_contents('test.txt', "Строка1\n", LOCK_EX);
// Добавление в конец
file_put_contents('test.txt', "Строка2", LOCK_EX | FILE_APPEND);
```

Прочитать данные из файла позволяют следующие функции:

- `fread(<Дескриптор>, <Длина в байтах>)` — позволяет прочитать из файла строку указанной длины. Если функции не удалось прочесть заданное число байтов, она возвратит то, что удалось прочитать. Вот пример чтения из файла:

```
@$file = fopen('test_file.txt', 'rb') or die('Ошибка');
if (flock($file, LOCK_SH)) { // Устанавливаем блокировку
   $s = fread($file, 100);   // Читаем данные из файла
   echo $s; // Строка
   flock($file, LOCK_UN);    // Снимаем блокировку
}
fclose($file);               // Закрываем файл
```

- `fgets(<Дескриптор>[, <Длина в байтах>])` — позволяет считывать из файла по одной строке за раз. Считывание будет выполняться до тех пор, пока не встретится символ новой строки (`\n`), конец файла или из файла не будет прочитано указанное число байтов. Если данных для чтения больше нет, функция вернет значение `false`:

```
@$file = fopen('test.txt', 'rb') or die('Ошибка');
while (($s = fgets($file, 4096)) !== false) {
   echo $s;
} // Строка1\nСтрока2
fclose($file);
```

- `fgetc(<Дескриптор>)` — считывает из файла один байт. Если данных для чтения больше нет, функция вернет значение `false`;
- `feof(<Дескриптор>)` — возвращает значение `true`, если был достигнут конец файла, и `false` — в противном случае;
- `file(<Путь к файлу>[, <Режим>[, <Контекст>]])` — читает весь файл в массив, каждый элемент которого будет равен одной строке, прочитанной из файла. В параметре `<Режим>` могут быть указаны следующие константы:
  - `FILE_USE_INCLUDE_PATH` — поиск файла будет осуществлен в каталогах, перечисленных в директиве `include_path`;
  - `FILE_IGNORE_NEW_LINES` — не добавлять символ новой строки в конец элемента массива;
  - `FILE_SKIP_EMPTY_LINES` — игнорировать пустые строки.

  Пример:

```
$arr = file('test.txt', FILE_IGNORE_NEW_LINES);
print_r($arr);
// Array ( [0] => Строка1 [1] => Строка2 )
```

- `readfile(<Путь к файлу>[, <true | false>[, <Контекст>]])` — открывает файл и выводит все его содержимое в окно Web-браузера. Если во втором параметре указано значение `true`, то поиск файла будет осуществлен в каталогах, перечисленных в директиве `include_path`:

```
readfile('test.txt'); // Строка1\nСтрока2
```

- `file_get_contents(<Путь к файлу>[, <true | false>[, <Контекст>[, <Начальная позиция>[, <Максимальная длина>]]]])` — возвращает содержимое файла в виде строки. В отличие от функции `readfile()` не выводит содержимое файла в окно Web-браузера. Если в параметре `<Флаг>` указано значение `true`, то поиск файла будет осуществлен в каталогах, перечисленных в директиве `include_path`.

```
$s = file_get_contents('test.txt');
echo htmlspecialchars($s, ENT_COMPAT | ENT_HTML5, 'UTF-8');
```

Для примера создадим файл file.txt и запишем в него пять адресов E-mail по одному на строчке (листинг 5.119).

Листинг 5.119. Создание файла и запись в него

```
<?php
$arr = ['mail1@site.ru', 'mail2@site.ru', 'mail3@site.ru',
        'mail4@site.ru', 'mail5@site.ru'];
$s = implode("\n", $arr);
@$file = fopen('file.txt', 'wb') or die('Ошибка');
if (flock($file, LOCK_EX)) { // Устанавливаем блокировку
   fwrite($file, $s);         // Записываем данные
   fflush($file);             // Сбрасываем буфер
   flock($file, LOCK_UN);     // Снимаем блокировку
}
else echo 'Не удалось установить блокировку';
fclose($file);                // Закрываем файл
echo 'Файл создан';
```

Если в процессе создания файла возникнет ошибка, то она будет подавлена оператором `@`, а в окне Web-браузера будет выведено сообщение **Ошибка**. При этом дальнейшая обработка файла будет остановлена.

Теперь добавим новую запись в конец файла (листинг 5.120).

Листинг 5.120. Добавление новой записи в конец файла

```
<?php
$mail = "\nmail6@site.ru";
@$file = fopen('file.txt', 'ab');
if (is_resource($file)) {
   if (flock($file, LOCK_EX)) {
      fwrite($file, $mail);
      fflush($file);
      flock($file, LOCK_UN);
   }
   else echo 'Не удалось установить блокировку';
   fclose($file);
   echo 'Операция произведена';
}
else echo 'Не удалось открыть файл';
```

В этом примере мы воспользовались функцией `is_resource()`, которая проверяет, содержит ли переменная идентификатор ресурса.

А теперь выведем содержимое файла в список (листинг 5.121).

Листинг 5.121. Вывод содержимого файла в список

```
<?php
@$file = fopen('file.txt', 'rb');
```

```
if (is_resource($file)) {
   if (flock($file, LOCK_SH)) {
      echo "<select>\n";
      while (($s = fgets($file, 4096)) !== false) {
         echo '<option>';
         echo htmlspecialchars(trim($s),
                              ENT_COMPAT | ENT_HTML5, 'UTF-8');
         echo "</option>\n";
      }
      echo "</select>\n";
      flock($file, LOCK_UN);
   }
   else echo 'Не удалось установить блокировку';
   fclose($file);
}
else echo 'Не удалось открыть файл';
```

Результат:

```
<select>
<option>mail1@site.ru</option>
<option>mail2@site.ru</option>
<option>mail3@site.ru</option>
<option>mail4@site.ru</option>
<option>mail5@site.ru</option>
<option>mail6@site.ru</option>
</select>
```

5.16.4. Перемещение указателя внутри файла

Для каждого открытого файла существует особый указатель, помечающий в нем текущую позицию. Изменить позицию указателя внутри файла можно с помощью следующих функций:

- `rewind(<Дескриптор>)` — устанавливает указатель на начало файла;
- `ftruncate(<Дескриптор>, <Размер>)` — обрезает файл до указанного размера;
- `ftell(<Дескриптор>)` — возвращает позицию указателя относительно начала файла;
- `fseek(<Дескриптор>, <Смещение>[, <Позиция>])` — устанавливает указатель в позицию, имеющую смещение `<Смещение>` относительно позиции `<Позиция>`. Параметр `<Позиция>` может принимать следующие значения:
  - `SEEK_SET` — начало файла (по умолчанию);
  - `SEEK_CUR` — текущая позиция указателя;
  - `SEEK_END` — конец файла.

ПРИМЕЧАНИЕ

При работе с кодировкой UTF-8 следует учитывать, что корректной будет только операция установки указателя в начало или в конец файла. Любое промежуточное значение может указывать на позицию внутри буквы, что при записи приведет к повреждению данных.

Пример перемещения указателя внутри файла приведен в листинге 5.122.

Листинг 5.122. Перемещение указателя внутри файла

```
<?php
@$file = fopen('file.txt', 'r+b');
fseek($file, 0, SEEK_END);   // В конец файла
var_dump(ftell($file));      // int(83)
rewind($file);               // В начало
var_dump(ftell($file));      // int(0)
fseek($file, 5, SEEK_CUR);   // 5 байтов от текущей позиции
var_dump(ftell($file));      // int(5)
fclose($file);
```

5.16.5. Создание списка рассылки с возможностью добавления, изменения и удаления адресов E-mail

В качестве примера работы с файлами рассмотрим создание списков рассылки. Включим возможность добавления нового E-mail, удаления и переименования существующих, а также выведем содержимое файла в поле <textarea>. Для этого создадим два файла: mail_script.inc (листинг 5.123) и mail.php (листинг 5.124).

Листинг 5.123. Содержимое файла C:\xampp\php\includes\mail_script.inc

```
<?php
// Проверка E-mail на корректность
function test_email($email) {
   $pattern = '/^([a-z0-9_.-]+)@([a-z0-9-]+\\.)+[a-z]{2,6}$/isu';
   return preg_match($pattern, $email);
}
// Проверка наличия E-mail. Возвращает индекс или -1
function index_email_in_array($email, &$arr) {
   foreach ($arr as $i => $value) {
      if (strtolower($email) === strtolower($value))
         return $i;
   }
   return -1;
}
```

```
// Чтение файла в массив
function read_file_to_array($path) {
   if (!file_exists($path)) { // Если файл не существует
      throw new Exception('Файл не существует');
   }
   @$arr = file($path,
                FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES);
   if (is_array($arr)) return $arr;
   else throw new Exception('Не удалось прочитать файл');
}
// Сохранение массива в файл
function save_array_to_file($path, &$arr) {
   sort($arr, SORT_NATURAL | SORT_FLAG_CASE);
   @$st = file_put_contents($path, implode("\n", $arr), LOCK_EX);
   if ($st === false)
      throw new Exception('Ошибка: не удалось записать в файл');
}
// Вывод содержимого файла
function show_emails($path) {
   echo '<textarea cols="25" rows="15">';
   if (file_exists($path)) readfile($path);
   else echo 'Файл не найден';
   echo '</textarea>';
}
// Замена специальных символов
function hsc($str) {
   return htmlspecialchars($str, ENT_COMPAT | ENT_HTML5, 'UTF-8');
}
// Сообщение о не соответствии E-mail шаблону
function bad_email_message($email) {
   $msg = '<div style="color: red">E-mail ' . hsc($email);
   $msg .= " не соответствует шаблону</div>\n";
   return $msg;
}
// Добавление E-mail
function add_email($path, $email) {
   if (!test_email($email)) {
      return [hsc($email), bad_email_message($email)];
   }
   $arr = read_file_to_array($path);
   if (index_email_in_array($email, $arr) == -1) {
      $arr[] = $email;
      save_array_to_file($path, $arr);
      $msg = '<div style="color: green">E-mail добавлен</div>';
      return ['', $msg];
   }
```

```
    else {
        $msg = '<div style="color: red">E-mail был добавлен ';
        $msg .= "ранее</div>\n";
        return [$email, $msg];
    }
}
// Удаление E-mail
function delete_email($path, $email) {
    if (!test_email($email)) {
        return [hsc($email), bad_email_message($email)];
    }
    $arr = read_file_to_array($path);
    $index = index_email_in_array($email, $arr);
    if ($index != -1) {
        unset($arr[$index]);
        save_array_to_file($path, $arr);
        $msg = '<div style="color: green">E-mail удален</div>';
        return ['', $msg];
    }
    else {
        $msg = '<div style="color: red">E-mail не найден</div>';
        return [hsc($email), $msg];
    }
}
// Изменение E-mail
function update_email($path, $old_email, $new_email) {
    if (!test_email($old_email)) {
        return [hsc($old_email), hsc($new_email),
                bad_email_message($old_email)];
    }
    if (!test_email($new_email)) {
        return [hsc($old_email), hsc($new_email),
                bad_email_message($new_email)];
    }
    $arr = read_file_to_array($path);
    $index = index_email_in_array($old_email, $arr);
    if ($index == -1) {
        $msg = '<div style="color: red">E-mail не найден</div>';
        return [hsc($old_email), hsc($new_email), $msg];
    }
    if (index_email_in_array($new_email, $arr) != -1) {
        $msg = '<div style="color: red">Добавляемый E-mail ';
        $msg .= 'зарегистрирован ранее</div>';
        return [hsc($old_email), hsc($new_email), $msg];
    }
    else {
        $arr[$index] = $new_email;
```

```
      save_array_to_file($path, $arr);
      $msg = '<div style="color: green">E-mail ';
      $msg .= 'изменен</div>';
      return ['', '', $msg];
   }
}
?>
```

Листинг 5.124. Содержимое файла mail.php

```
<?php
require_once 'HTML5Header.inc';    // Листинг 5.88
require_once 'mail_script.inc';    // Листинг 5.123
(new HTML5Header('Создание списка рассылки', ''))->show();
echo "<div>\n";
// Путь к файлу
$path = 'file.txt';

if (isset($_GET['add'])) {
   $add = $_GET['add'];
   try {
      list($add, $msg) = add_email($path, $add);
   } catch (Exception $e) {
      $msg = '<div style="color: red">';
      $msg .= "{$e->getMessage()}</div>\n";
      $add = '';
   }
   echo $msg;
}
else $add = '';
?>
<!-- Выводим форму Добавить -->
<form action="<?=$_SERVER['SCRIPT_NAME']?>">
<input type="text" name="add" value="<?=$add?>">
<input type="submit" value="Добавить">
</form>
<?php
if (isset($_GET['del'])) {
   $del = $_GET['del'];
   try {
      list($del, $msg) = delete_email($path, $del);
   } catch (Exception $e) {
      $msg = '<div style="color: red">';
      $msg .= "{$e->getMessage()}</div>\n";
      $del = '';
   }
```

```
   echo $msg;
}
else $del = '';
?>
<!-- Выводим форму Удалить -->
<form action="<?=$_SERVER['SCRIPT_NAME']?>">
<input type="text" name="del" value="<?=$del?>">
<input type="submit" value="Удалить">
</form>
<?php
if (isset($_GET['old_email']) && isset($_GET['new_email'])) {
   $old_email = $_GET['old_email'];
   $new_email = $_GET['new_email'];
   try {
      list($old_email, $new_email, $msg) =
           update_email($path, $old_email, $new_email);
   } catch (Exception $e) {
      $msg = '<div style="color: red">';
      $msg .= "{$e->getMessage()}</div>\n";
      $old_email = $new_email = '';
   }
   echo $msg;
}
else $old_email = $new_email = '';
?>
<!-- Выводим форму Изменить -->
<form action="<?=$_SERVER['SCRIPT_NAME']?>">
<input type="text" name="old_email" value="<?=$old_email?>"
       placeholder="Старый E-mail">
<input type="text" name="new_email" value="<?=$new_email?>"
       placeholder="Новый E-mail">
<input type="submit" value="Изменить">
</form>
<!-- Выводим содержимое файла -->
<?php show_emails($path); ?>
</div>
</body>
</html>
```

Откроем в Web-браузере файл mail.php. С помощью предлагаемых им форм можно добавить новый E-mail, а также удалить или переименовать существующий. Причем добавить можно только новый E-mail — если будет введен уже существующий E-mail, то в Web-браузере отобразится соответствующее предупреждение. Кроме того, выполняется проверка на корректность введенного E-mail — если он не соответствует шаблону, то также отобразится сообщение. Заменить E-mail можно только на E-mail, отсутствующий в файле. Таким образом, в файле будут записаны лишь уникальные адреса E-mail.

Как разослать письма по адресам E-mail из этого файла, мы рассмотрим при изучении отправки писем с сайта (см. *разд. 5.17.16*).

5.16.6. Чтение CSV-файлов. Преобразование CSV-файла в HTML-таблицу

При работе с таблицами (например, в Excel) есть возможность сохранения таблицы в формате CSV. В этом формате каждая строка будет содержать значения ряда ячеек таблицы, разделенных точкой с запятой.

Например, таблица:

1	2	3	4
5	6	7	8
9	10	11	12

при сохранении в формате CSV будет выглядеть следующим образом:

```
1;2;3;4
5;6;7;8
9;10;11;12
```

Для чтения CSV-файлов предусмотрена функция `fgetcsv()`. Формат функции:

```
fgetcsv(<Дескриптор>[, <Длина в байтах>[, <Разделитель>[,
        <Ограничитель>[, <Экранирование>]]]])
```

Если <Разделитель> не указан, то по умолчанию используется символ , (запятая). Если не указан <Ограничитель>, то по умолчанию используется символ " (кавычка). Если не определен параметр <Экранирование>, то по умолчанию используется символ \ (слеш).

Функция `fgetcsv()` считывает из файла одну строку при каждом вызове. Считывание будет выполняться до тех пор, пока не встретится символ новой строки (\n), конец файла или из файла не будет прочитано указанное число байтов. Строка будет разбита по разделителю <Разделитель> и помещена в возвращаемый массив.

Если какая-либо ячейка содержит символ разделителя, то все содержимое ячейки обычно заключается в кавычки. Если используется другой символ, то он должен быть указан в параметре <Ограничитель>.

Таким образом, при сохранении в формате CSV таблица следующего вида:

1	2	3	4
5	6	7	8
9	10	11	12;15

будет выглядеть так:

```
1;2;3;4
5;6;7;8
9;10;11;"12;15"
```

Чтобы преобразовать CSV-файл в HTML-таблицу, можно воспользоваться кодом, приведенным в листинге 5.125.

Листинг 5.125. Преобразование CSV-файла в HTML-таблицу

```
<?php
require_once 'HTML5Header.inc'; // Листинг 5.88
$title = 'Преобразование CSV-файла в HTML-таблицу';
$meta = '<style>table { border: black 1px solid; border-spacing: 0 }';
$meta .= ' td { border: black 1px solid; padding: 5px }</style>';
(new HTML5Header($title, '', $meta))->show();
@$file = fopen('filecsv.csv', 'rb') or die('Ошибка');
flock($file, LOCK_SH);
echo '<table>', "\n";
while(($data = fgetcsv($file, 1024, ';')) !== false) {
   $count = count($data);
   if ($count > 0) {
      echo "<tr>\n";
      for ($i = 0; $i < $count; $i++) {
         echo '<td>';
         echo htmlspecialchars($data[$i], ENT_COMPAT | ENT_HTML5,
                               'UTF-8');
         echo "</td>\n";
      }
      echo "</tr>\n";
   }
}
echo '</table>';
flock($file, LOCK_UN);
fclose($file);
echo '</body></html>';
```

Записать в файл строку в формате CSV позволяет функция `fputcsv()`. После записи вставляется символ перевода строки. Формат функции:

```
fputcsv(<Дескриптор>, <Массив>[, <Разделитель>=','[,
        <Ограничитель>='"'[, <Экранирование>='\']]])
```

Пример:

```
@$file = fopen('filecsv.txt', 'wb');
$data = ['1', '2', '3', '4, 5'];
fputcsv($file, $data, ';', '"');
fclose($file);
```

Строка в файле будет выглядеть так:

```
1;2;3;"4, 5"
```

5.16.7. Права доступа в операционной системе UNIX

Большинство хостинговых площадок используют операционные системы семейства UNIX. В этой ОС каждому объекту (файлу или каталогу) назначаются права доступа для каждой разновидности пользователей: владельца, группы и прочих. Рассмотрим эту важную тему подробнее.

Для файла или каталога могут быть назначены следующие права доступа:

- чтение;
- запись;
- выполнение.

Права доступа обозначаются буквами:

- `r` — файл можно читать, а содержимое каталога можно просматривать;
- `w` — файл можно модифицировать, удалять и переименовывать, а в каталоге можно создавать или удалять файлы. Каталог можно переименовать или удалить;
- `x` — файл можно выполнять, а в каталоге можно выполнять операции над файлами, в том числе искать в нем файлы.

Права доступа к файлу определяются записью типа:

```
-rw-r--r--
```

Первый символ означает, что это файл, и не задает никаких прав доступа. Следующие три символа (`rw-`) задают права доступа для владельца: чтение и запись. Символ `-` здесь означает, что права доступа на выполнение нет. Следующие три символа (`r--`) задают права доступа для группы: только чтение. Последние три символа (`r--`) задают права для всех остальных пользователей: только чтение.

Права доступа к каталогу определяются такой строкой:

```
drwxr-xr-x
```

Первая буква (`d`) означает, что это каталог. Владелец может выполнять в каталоге любые действия (`rwx`), а группа и все остальные пользователи — только читать и выполнять поиск (`r-x`). Для того чтобы каталог можно было просматривать, должны быть установлены права на выполнение (`x`).

Кроме того, права доступа могут обозначаться числом. Такие числа называются *маской прав доступа*. Число состоит из трех цифр: от `0` до `7`. Первая цифра задает права для владельца, вторая — для группы, а третья — для всех остальных пользователей. Например, права доступа `-rw-r--r--` соответствуют числу `644`.

Сопоставим числам, входящим в маску прав доступа, двоичную и буквенную записи (табл. 5.1).

Например, права доступа `rw-r--r--` можно записать так: `110 100 100`, что переводится в число `644`. Таким образом, если право предоставлено, то в соответствующей позиции стоит `1`, а если нет — то `0`.

Таблица 5.1. Права доступа в разных записях

Восьмеричная цифра	Двоичная запись	Буквенная запись
0	000	---
1	001	--x
2	010	-w-
3	011	-wx
4	100	r--
5	101	r-x
6	110	rw-
7	111	rwx

Для файлов, не являющихся CGI-программами (таких как *.html, *.shtml или *.php), права доступа могут быть установлены равными 644 (rw-r--r--) — чтение и запись для владельца и только чтение для всех остальных.

Для файлов, являющихся CGI-программами (Perl-скрипты, скомпилированные программы на языке C и прочие), права доступа должны быть установлены в 755 (rwxr-xr-x) — чтение, запись и исполнение для владельца и чтение и исполнение для всех остальных.

Если Web-сервер запускает PHP-скрипты от имени владельца, то для записи данных в файл вполне достаточно поставить на этот файл права доступа 600 (rw-------) — если, конечно, файл не предназначен для чтения всеми пользователями.

Права доступа на каталоги рекомендуется устанавливать в 755 (rwxr-xr-x).

Чтобы изменить права доступа из скрипта, необходимо воспользоваться функцией chmod(). Формат функции:

```
chmod(<Путь к файлу>, <Права доступа>)
```

Права доступа задаются в виде числа, перед которым следует указать 0 (это соответствует восьмеричной записи числа):

```
chmod($path, 0644);
```

Определить права доступа можно с помощью следующих функций:

- is_readable(<Путь>) — возвращает true, если файл доступен для чтения;
- is_writable(<Путь>) — возвращает true, если файл доступен для записи;
- is_executable(<Путь>) — возвращает true, если файл является выполняемым.

5.16.8. Функции для работы с файлами

Рассмотрим основные функции для работы с файлами.

- `copy(<Копируемый файл>, <Куда копируем>[, <Контекст>])` — позволяет скопировать файл. Если файл существует, то он будет перезаписан. Функция возвращает `true`, если файл удачно скопирован:

```
if (@copy('test.txt', 'test2.txt')) echo 'Файл скопирован';
```

- `rename(<Старое имя>, <Новое имя>[, <Контекст>])` — переименовывает файл или каталог. Если файл с новым именем существует, то он будет перезаписан. Если файл переименован, то функция вернет `true`:

```
if (@rename('test2.txt', 'test3.txt')) echo 'Файл переименован';
```

- `unlink(<Путь>[, <Контекст>])` — позволяет удалить файл. Функция вернет `true`, если файл был удален:

```
if (@unlink('test3.txt')) echo 'Файл удален';
```

- `file_exists(<Путь>)` — проверяет наличие файла или каталога. Значением функции будет `true`, если файл найден:

```
if (file_exists('test.txt')) echo 'Файл существует';
```

- `is_file(<Объект>)` — возвращает `true`, если объект является файлом:

```
if (is_file('test.txt')) echo 'Файл';
```

- `basename(<Путь>[, <Суффикс>])` — возвращает имя файла без пути к нему:

```
echo basename('C:/xampp/htdocs/test.txt');
// Выведет: test.txt
echo basename('C:/xampp/htdocs/test.txt', '.txt');
// Выведет: test
```

- `dirname(<Путь>[, <Уровень>])` — возвращает путь к каталогу:

```
echo dirname('C:/xampp/htdocs/test.txt');
// Выведет: C:/xampp/htdocs
echo dirname('C:/xampp/htdocs/test.txt', 2);
// Выведет: C:/xampp
```

- `realpath(<Относительный путь к файлу>)` — преобразует относительный путь к файлу в абсолютный:

```
echo realpath('test.txt');
// Выведет: C:\xampp\htdocs\test.txt
```

  Обратите внимание: функция также проверяет наличие файла, и если файла нет, то функция вернет значение `false`;

- `filesize(<Путь к файлу>)` — возвращает размер файла:

```
echo filesize('test.txt');
```

- fileatime(<Путь к файлу>) — служит для определения времени последнего доступа к файлу:

```
$date = date('d-m-Y', fileatime('index.php'));
echo $date; // Выведет: 03-01-2018
```

- filectime(<Путь к файлу>) — позволяет узнать время создания файла:

```
$date = date('d-m-Y', filectime('index.php'));
echo $date; // Выведет: 03-01-2018
```

- filemtime(<Путь к файлу>) — возвращает время последнего изменения файла:

```
$date = date('d-m-Y', filemtime('index.php'));
echo $date; // Выведет: 06-01-2018
```

- touch($filename[, $time[, $atime]]) — устанавливает для файла время последнего изменения. Если параметры $time и $atime не указаны, то используется текущее время. Если файла нет, то он будет создан:

```
touch('index.php');
```

- stat(<Путь к файлу>) и fstat(<Дескриптор>) — возвращают подробную информацию о файле в виде массива:

```
var_dump(stat('index.php'));
@$file = fopen('index.php', 'rb');
var_dump(fstat($file));
fclose($file);
```

5.16.9. Загрузка файлов на сервер

Загрузка файлов на сервер осуществляется с помощью формы, у которой параметр enctype равен multipart/form-data. Создадим файл file_load.html с содержимым, приведенным в листинге 5.126.

Листинг 5.126. Содержимое файла file_load.html

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Загрузка файлов</title>
</head>
<body>
 <h1>Загрузка файлов</h1>
 <form action="file.php" method="POST" enctype="multipart/form-data">
  <div>
    <input type="file" name="file_name">
    <input type="submit" value="Загрузить">
  </div>
```

```
 </form>
</body>
</html>
```

Далее создаем файл file.php и добавляем в него код, приведенный в листинге 5.127.

Листинг 5.127. Содержимое файла file.php

```
<?php
if (isset($_FILES['file_name'])) {
   if ($_FILES['file_name']['error'] === 0 &&
       $_FILES['file_name']['size'] > 0) {
         $path = "C:\\xampp\\htdocs\\";
         $path .= basename($_FILES['file_name']['name']);
         if (@move_uploaded_file($_FILES['file_name']['tmp_name'],
                                 $path)) {
            echo 'Файл успешно загружен';
         }
         else {
            echo 'Ошибка при загрузке';
         }
      }
      else echo 'Ошибка при загрузке';
}
print_r($_FILES);
```

В адресной строке Web-браузера набираем: `http://localhost/file_load.html`. При выборе файла с помощью кнопки **Обзор** и нажатии кнопки **Загрузить** выбранный файл будет отправлен серверу. Например, мы отправляем файл foto.jpg. Получив файл, сервер сохраняет его в каталоге для временных файлов и создает в суперглобальном массиве `$_FILES` следующие элементы:

```
Array (
    [file_name] => Array
        (
            [name] => foto.jpg
            [type] => image/jpeg
            [tmp_name] => C:\xampp\tmp\phpAECA.tmp
            [error] => 0
            [size] => 94967
        )
)
```

Значение `file_name` здесь может изменяться — это название поля выбора файла в HTML-форме, а остальные параметры неизменны, и соответствующие им элементы ассоциативного массива содержат следующие данные:

- `name` — первоначальное название файла;
- `type` — MIME-тип файла;

- tmp_name — путь и название временного файла;
- size — размер файла;
- error — код ошибки. Может принимать следующие значения:
  - 0 — UPLOAD_ERR_OK — ошибок нет, файл был успешно загружен на сервер;
  - 1 — UPLOAD_ERR_INI_SIZE — размер принятого файла превысил максимально допустимую величину, которая задана директивой upload_max_filesize конфигурационного файла php.ini;
  - 2 — UPLOAD_ERR_FORM_SIZE — размер загружаемого файла превысил значение MAX_FILE_SIZE, указанное в скрытом поле HTML-формы;
  - 3 — UPLOAD_ERR_PARTIAL — загружаемый файл был получен только частично;
  - 4 — UPLOAD_ERR_NO_FILE — файл не загружен;
  - 6 — UPLOAD_ERR_NO_TMP_DIR — отсутствует временный каталог;
  - 7 — UPLOAD_ERR_CANT_WRITE — не удалось записать файл на диск;
  - 8 — UPLOAD_ERR_EXTENSION — PHP-расширение остановило загрузку файла.

Итак, файл загружен в каталог временных файлов. Теперь необходимо проверить, не возникло ли проблем с загрузкой. Если все в порядке, то переменная окружения $_FILES['file_name']['error'] будет содержать значение 0. Затем нужно скопировать файл из каталога временных файлов в нужный каталог. Если файл не скопировать из каталога временных файлов, то по завершении работы сценария он будет удален. Скопировать файл позволяет функция move_uploaded_file(). Формат функции:

```
move_uploaded_file(<Загруженный файл>, <Куда копируем>)
```

Функция перемещает загруженный файл в новое место, первоначально проверяя, является ли файл загруженным на сервер (переданным по протоколу POST). Если файл действительно загружен на сервер, он будет перемещен в место, указанное во втором параметре. Если он не является загруженным файлом, никаких действий не предпринимается, и функция возвращает false. Если файл, указанный во втором параметре, уже существует, он будет перезаписан. Если файл был успешно перемещен, то функция возвратит true.

Если нужно загрузить сразу несколько файлов, то в HTML-форме после имени поля следует указать квадратные скобки (признак массива в PHP):

```
<input type="file" name="file_name[]">
<input type="file" name="file_name[]">
<input type="file" name="file_name[]">
```

В этом случае суперглобальный массив $_FILES будет иметь следующую структуру (показан лишь фрагмент с ключом name):

```
Array (
    [file_name] => Array (
```

```
            [name] => Array
                (
                    [0] => foto.jpg
                    [1] => foto2.jpg
                    [2] => foto3.jpg
                )
            ... Фрагмент опущен
        )
)
```

Таким образом, после имени переменной окружения нужно дополнительно указать индекс файла: `$_FILES['file_name']['tmp_name'][0]`. Все остальное точно так же, но следует учитывать совокупный размер загружаемых файлов, который может быть превышен при загрузке сразу нескольких файлов.

5.16.10. Функции для работы с каталогами

Для работы с каталогами предусмотрены следующие функции:

- `mkdir(<Путь>[, <Права доступа>=0777[, <true |false>=false[, <Контекст>]]])` — создает новый каталог с правами доступа, указанными во втором параметре. Права доступа указываются в виде трехзначного числа, перед которым ставится `0`, — например, `0755`. Если в третьем параметре указано значение `true`, то будут созданы все каталоги в пути. Если каталог создан, функция вернет значение `true`, в противном случае — `false`:

```
if (@mkdir('folder1', 0755)) echo 'Каталог создан';
if (@mkdir('folder2/filder3', 0755, true))
    echo 'Каталоги созданы';
```

- `rmdir(<Путь>[, <Контекст>])` — удаляет пустой каталог. Если в каталоге есть файлы, то каталог удален не будет. Если каталог удален, функция вернет значение `true`, в противном случае — `false`:

```
if (@rmdir('folder1')) echo 'Каталог удален';
```

- `rename(<Старое имя>, <Новое имя>[, <Контекст>])` — переименовывает файл или каталог. Если каталог переименован, то функция вернет `true`:

```
if (@rename('folder2', 'folder')) echo 'Каталог переименован';
```

- `getcwd()` — возвращает текущий рабочий каталог:

```
print_r(getcwd()); // C:\xampp\htdocs
```

- `chdir(<Путь>)` — делает указанный каталог текущим:

```
chdir('C:\\xampp');
print_r(getcwd()); // C:\xampp
```

- `opendir(<Путь>[, <Контекст>])` — открывает каталог для чтения. Функция возвращает дескриптор, который указывается в других функциях, или `false` — в случае ошибки;

- readdir(<Дескриптор>) — считывает следующее имя объекта (файла или подкаталога);
- closedir(<Дескриптор>) — закрывает каталог:

```
@$fd = opendir('C:\\xampp\\htdocs');
if ($fd) {
   while (($obj = readdir($fd)) !== false) {
      $obj = htmlspecialchars($obj,
                 ENT_COMPAT | ENT_HTML5, 'UTF-8');
      echo "$obj<br>\n";
   }
   closedir($fd);
}
```

- rewinddir(<Дескриптор>) — перемещает указатель в начало каталога;
- is_dir(<Объект>) — возвращает true, если объект является каталогом;
- is_file(<Объект>) — возвращает true, если объект является файлом;
- is_link(<Объект>) — возвращает true, если объект является символической ссылкой;
- scandir(<Путь>[, <Сортировка>[, <Контекст>]]) — возвращает содержимое каталога в виде массива. Во втором параметре можно указать константы SCANDIR_SORT_NONE (без сортировки), SCANDIR_SORT_ASCENDING (сортировка по возрастанию, значение по умолчанию) и SCANDIR_SORT_DESCENDING (сортировка по убыванию):

```
$arr = scandir('C:\\xampp\\htdocs', SCANDIR_SORT_NONE);
print_r($arr);
$arr = scandir('C:\\xampp\\htdocs', SCANDIR_SORT_ASCENDING);
print_r($arr);
$arr = scandir('C:\\xampp\\htdocs', SCANDIR_SORT_DESCENDING);
print_r($arr);
```

- glob(<Шаблон>[, <Флаги>]) — возвращает массив с объектами, соответствующими шаблону. В шаблоне можно указать следующие специальные символы:
  - ? — любой одиночный символ;
  - * — любое число любых символов;
  - [<Символы>] — позволяет указать символы, которые должны быть на этом месте в пути. Можно вписать символы или указать диапазон через тире.

  В параметре <Флаги> можно указать комбинацию констант GLOB_MARK, GLOB_NOSORT (без сортировки), GLOB_NOCHECK, GLOB_NOESCAPE, GLOB_BRACE, GLOB_ONLYDIR (только каталоги) и GLOB_ERR (остановиться при ошибке). Вот пример получения всех файлов с расширением php:

```
$arr = glob('*.php');
print_r($arr);
```

- `DIRECTORY_SEPARATOR` — эта константа содержит символ-разделитель в пути:

```
print_r(DIRECTORY_SEPARATOR); // Выведет в Windows: \
```

5.16.11. Создаем программу для просмотра всех доступных каталогов и файлов на диске

Прочитаем содержимое каталога C:\xampp\htdocs и выведем его содержимое в окно Web-браузера. Каталоги и файлы выведем отдельно, а для файлов укажем размер, дату создания и дату изменения файла. Кроме того, добавим возможность перемещения по файловой системе с помощью гиперссылок и предусмотрим возможность использования русских букв в названиях каталогов и файлов. Для этого создадим два файла: dir_script.inc (листинг 5.128) и dir.php (листинг 5.129).

Листинг 5.128. Содержимое файла C:\xampp\php\includes\dir_script.inc

```
<?php
function new_url($path) {
   $arr = explode('/', $path);
   if (count($arr) > 1) {
      array_pop($arr); // Удаляем последний элемент
      return implode('/', $arr);
   }
   else return '';
}
function read_dir($path, &$d, &$f) {
   @$fd = opendir($path); // Открываем каталог
   if ($fd) {
      chdir($path); // Делаем каталог текущим
      while (($obj = readdir($fd)) !== false) {
         if (is_dir($obj)) { // Если это каталог
            if ($obj != '.') {
               $d[] = $obj;
            }
         }
         elseif (is_file($obj)) { // Если это файл
            $size = filesize($obj);
            $cdate = date('d-m-Y', filectime($obj));
            $mdate = date('d-m-Y', filemtime($obj));
            $f[] = array($obj, $size, $cdate, $mdate);
         }
      }
      closedir($fd); // Закрываем каталог
   }
   else exit('Не удалось открыть каталог');
}
```

```
// Замена специальных символов
function hsc($str) {
   return htmlspecialchars($str, ENT_COMPAT | ENT_HTML5, 'UTF-8');
}
?>
```

Листинг 5.129. Содержимое файла dir.php

```
<?php
require_once('HTML5Header.inc');    // Листинг 5.88
require_once('dir_script.inc');     // Листинг 5.128
// Выводим заголовок
$style =<<<LABEL
   <style>
      table { border-spacing: 0; width: 100% }
      td { padding: 5px }
      h2 { text-align: center }
      .file_info tr:nth-child(2n) { background: #e8e8e8 }
      .file_info { border: black 1px solid; border-collapse: collapse }
      .file_info td, .file_info th { border: black 1px solid }
   </style>
LABEL;
(new HTML5Header('Просмотр каталогов', '', $style))->show();

$directory = array();
$files = array();
// Задаем путь по умолчанию
if (!isset($_GET['path'])) $path = 'C:/xampp/htdocs';
else $path = $_GET['path'];
if (strlen($path) == 0) exit('Не задан путь');
// Получаем файлы и папки текущего каталога
read_dir($path, $directory, $files);
$path2 = new_url($path);
// Кодируем все спецсимволы
$path = urlencode($path);
$path2 = urlencode($path2);
// Выводим содержимое каталога
?>
<table>
<tr><td style="width: 25%">
<h2>Каталоги</h2>
</td><td>
<h2>Файлы</h2>
</td></tr>
<tr><td style="vertical-align: top">
<?php
```

```
for ($i = 0, $c = count($directory); $i < $c; $i++) {
   if ($directory[$i] == '..') {
      echo '<a href="?path=' . $path2 . '">На уровень выше</a><br><br>';
   }
   else {
      echo '<a href="?path=' . $path . urlencode('/' . $directory[$i]);
      echo '">' . hsc($directory[$i]) . "</a><br>\n";
   }
}
?>
</td><td style="vertical-align: top">
<table class="file_info">
<tr style="text-align: center">
<th style="width: 40%">Название файла</th>
<th style="width: 20%">Размер файла</th>
<th style="width: 20%">Дата создания файла</th>
<th style="width: 20%">Дата последнего изменения</th>
</tr>
<?php
// Выводим названия файлов
for ($i = 0, $c = count($files); $i < $c; $i++) {
   echo '<tr style="text-align: center">', "\n";
   echo '<td>' . hsc($files[$i][0]) . "</td>\n";
   echo '<td>' . hsc($files[$i][1]) . "</td>\n";
   echo '<td>' . hsc($files[$i][2]) . "</td>\n";
   echo '<td>' . hsc($files[$i][3]) . "</td>\n";
   echo "</tr>\n";
}
echo "</table>\n";
if (count($files) == 0) {
   echo '<div style="text-align: center"><b>Нет файлов</b></div>';
}
?>
</td></tr></table>
</body></html>
```

Откроем в Web-браузере файл dir.php. В окне браузера отобразится содержимое каталога C:\xampp\htdocs. С помощью имеющихся гиперссылок можно перемещаться между каталогами, отображая их содержимое, — почти как в программе Проводник в Windows.

5.17. Взаимодействие с Интернетом

В предыдущих разделах мы научились работать с локальным сервером, на котором расположена программа. Однако скрипт может взаимодействовать и с другими серверами, используя различные протоколы передачи данных. Например, по прото-

колу HTTP мы можем получить данные о погоде или курсе валют с другого сайта, по протоколу FTP — загрузить файлы на сервер и т. д.

5.17.1. Диалог между Web-браузером и сервером

Как вы уже знаете, данные формы могут быть отправлены либо методом `GET`, либо методом `POST`. При отправке по методу `GET` данные формы пересылаются путем их добавления к URL-адресу после знака `?`. При отправке по методу `POST` данные передаются после всех HTTP-заголовков. Рассмотрим диалог между Web-браузером и сервером более подробно.

Предположим, у нас есть форма на странице **http://localhost/index.html**:

```
<form action="test.php" method="GET">
<input type="text" name="text1">
<input type="submit" value="Отправить">
</form>
```

При заполнении текстового поля и нажатии кнопки **Отправить** Web-браузер посылает следующий запрос:

```
GET /test.php?text1=Tekst+v+pole HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 6.2; Win64; x64; rv:56.0)
   Gecko/20100101 Firefox/56.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: ru-RU,ru;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://localhost/index.html
Connection: keep-alive
Upgrade-Insecure-Requests: 1
```

Обратите внимание на первую строку запроса. В ней первое слово обозначает метод передачи данных. В нашем случае это метод `GET`. Далее следует строка запроса:

```
/test.php?text1=Tekst+v+pole
```

Здесь указывается путь от корня сайта к файлу-обработчику (`test.php`). После знака вопроса передается имя поля и его значение (`text1=Tekst+v+pole`). За строкой запроса идет название протокола (`HTTP/1.1`). Доменное имя Web-сайта передается в заголовке `Host` без указания протокола.

Кроме этого, в запросе дополнительно указываются предпочитаемые MIME-типы (заголовок `Accept`), поддерживаемые языки (заголовок `Accept-Language`), методы сжатия (заголовок `Accept-Encoding`), информация о самом Web-браузере (заголовок `User-Agent`) и пр.

Если изменить метод передачи с `GET` на `POST`, то запрос станет иным:

```
POST /test.php HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 6.2; Win64; x64; rv:56.0)
   Gecko/20100101 Firefox/56.0
```

```
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: ru-RU,ru;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://localhost/index.html
Content-Type: application/x-www-form-urlencoded
Content-Length: 18
Connection: keep-alive
Upgrade-Insecure-Requests: 1

text1=Tekst+v+pole
```

В первой строке указывается метод передачи (`POST`), путь к файлу-обработчику от корня сайта (`test.php`) и название протокола (`HTTP/1.1`). Сами данные формы передаются после всех заголовков. Обратите внимание: данные формы от заголовков отделяет пустая строка. Длина переданных данных указывается в заголовке `Content-Length`, а метод их кодирования — в заголовке `Content-Type`.

На этот запрос Web-сервер посылает следующий ответ:

```
HTTP/1.1 200 OK
Date: Fri, 05 Jan 2018 21:43:56 GMT
Server: Apache/2.4.29 (Win32) OpenSSL/1.1.0g PHP/7.2.0
X-Powered-By: PHP/7.2.0
Content-Length: 66
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=UTF-8
Content-Language: ru
```

В первой строке ответа сервера указывается название протокола (`HTTP/1.1`), а затем статус ответа (`200`) и его текстовое описание (`OK`). Статус `200` указывает, что запрос был успешно обработан. Приведем основные коды статуса:

- `200` — запрос успешно обработан;
- `301` и `302` — перенаправление на другую страницу;
- `304` — с момента последнего запроса файл не изменялся;
- `401` — пользователь не авторизован;
- `403` — нет доступа. При отсутствии индексного файла в каталоге и отключенной опции `Indexes` директивы `Options` генерируется именно этот код;
- `404` — ресурс не найден;
- `500` — внутренняя ошибка сервера.

В заголовке `Content-Type` указываются MIME-тип (`text/html`) и кодировка передаваемых данных (`UTF-8`). С помощью заголовка `Content-Length` указывается длина передаваемых данных. Сами данные передаются после всех заголовков. Данные от заголовков отделяет пустая строка.

Чтобы увидеть диалог Web-браузера с сервером, воспользуемся панелью **Инструменты разработчика** Web-браузера Firefox. Открываем панель, нажав комбина-

цию клавиш <Shift>+<F2>, щелкаем на значке с ключом левой кнопкой мыши и переходим на вкладку **Сеть**. Можно также сразу нажать комбинацию клавиш <Ctrl>+<Shift>+<E>. Обновляем страницу и щелкаем мышью на строке запроса. Результат можно увидеть на рис. 5.1. Чтобы посмотреть заголовки без обработки, нажимаем кнопку **Необработанные заголовки**.

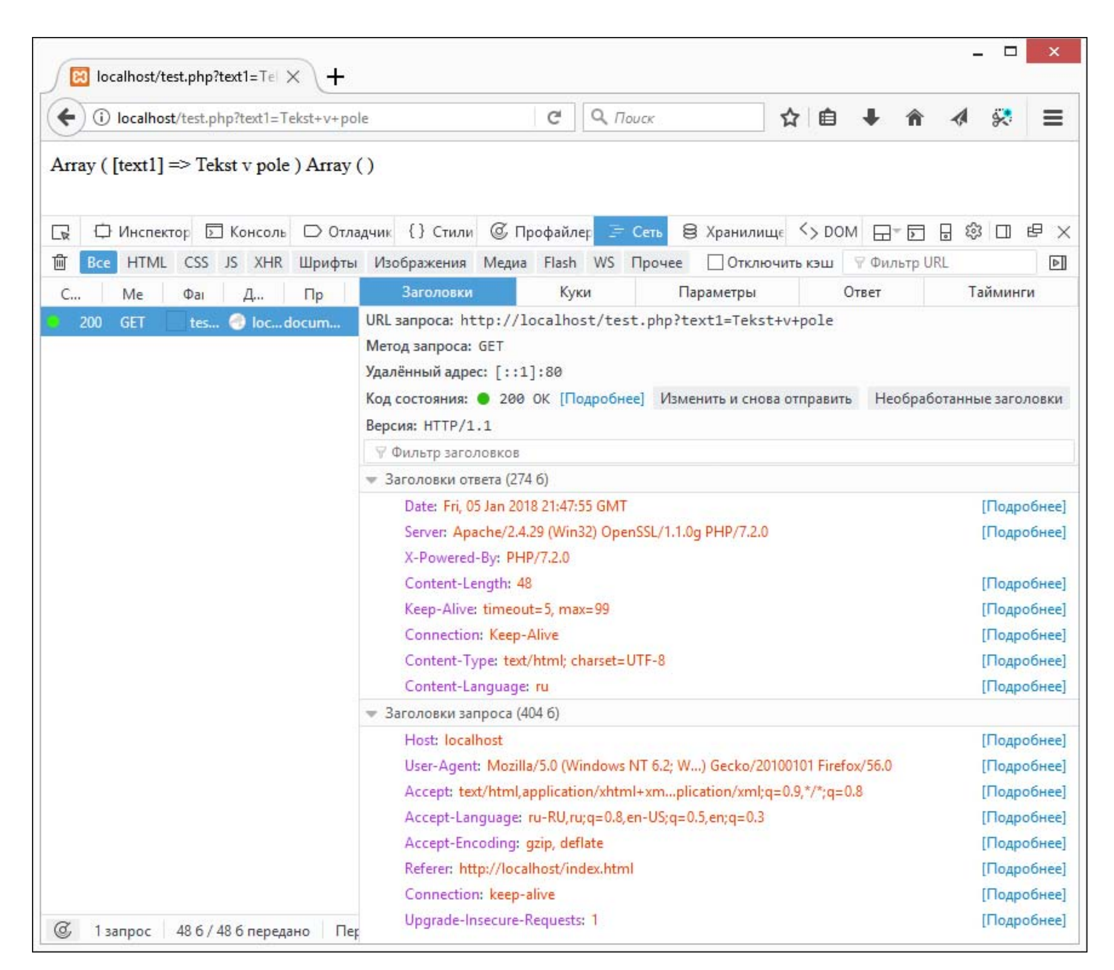


Рис. 5.1. Просмотр заголовков в Web-браузере Firefox

5.17.2. Основные заголовки HTTP

Заголовки HTTP предназначены для передачи некоторых дополнительных сведений, например, при запросе файла Web-браузером дополнительно указываются предпочитаемые MIME-типы, поддерживаемые языки и кодировки, информация о самом Web-браузере и т. д. Сервер в свою очередь при выдаче файла указывает MIME-тип файла, дату последней модификации файла, сведения о кодировке, языке и т. д.

Приведем основные заголовки:

- `Accept` — MIME-типы, предпочтительные для Web-браузера:

  ```
  Accept: text/html,application/xhtml+xml,application/xml;
  q=0.9,*/*;q=0.8
  ```

- `Accept-Language` — список предпочтительных для Web-браузера языков:

  ```
  Accept-Language: ru-RU,ru;q=0.8,en-US;q=0.5,en;q=0.3
  ```

- `Accept-Encoding` — список поддерживаемых Web-браузером методов сжатия:

  ```
  Accept-Encoding: gzip, deflate
  ```

- `Content-Type` — тип передаваемых данных:

  ```
  Content-Type: text/html; charset=UTF-8
  ```

- `Content-Length` — длина передаваемых данных при методе `POST` и длина ответа сервера:

  ```
  Content-Length: 18
  ```

- `Cookie` — информация об установленных cookies (отправляется Web-браузером):

  ```
  Cookie: n=12
  ```

- `Set-Cookie` — запрос на установку cookies (отправляется сервером):

  ```
  Set-Cookie: n=12
  ```

- `GET` — заголовок запроса при передаче данных методом `GET`;
- `POST` — заголовок запроса при передаче данных методом `POST`;
- `Host` — интернет-адрес хоста:

  ```
  Host: localhost
  ```

- `Last-Modified` — дата последней модификации файла:

  ```
  Last-Modified: Thu, 19 Oct 2017 01:19:33 GMT
  ```

- `Location` — перенаправление: при наличии этого заголовка Web-браузер обязан перейти по указанному URL-адресу:

  ```
  Location: http://localhost/firm.php
  ```

- `Pragma` — заголовок, управляющий кэшированием документа:

  ```
  Pragma: no-cache
  ```

- `Referer` — содержит URL-адрес, с которого пользователь перешел на наш сайт:

  ```
  Referer: http://localhost/index.html
  ```

- `Server` — содержит название и версию программного обеспечения сервера:

  ```
  Server: Apache/2.4.29 (Win32) OpenSSL/1.1.0g PHP/7.2.0
  ```

- `User-Agent` — содержит информацию об используемом Web-браузере:

  ```
  User-Agent: Mozilla/5.0 (Windows NT 6.2; Win64; x64; rv:56.0)
  Gecko/20100101 Firefox/56.0
  ```

Некоторые из этих заголовков Web-браузер посылает серверу (например, `Referer` и `User-Agent`), другие используются в ответе сервера (например, `Pragma` или `Server`), третьи могут пересылаться в обоих направлениях (скажем, `Content-Length`).

5.17.3. Функция *header()*

Функция `header()` позволяет добавить заголовок из программы. Формат функции:

```
header(string $string[, bool $replace=true[, int $code]]) : void
```

В первом параметре указывается заголовок HTTP:

```
<?php
header('Last-Modified: Thu, 19 Oct 2017 01:19:33 GMT');
```

Если во втором параметре указано значение `true`, то предыдущий заголовок будет заменен. Значение `false` позволяет указать сразу несколько однотипных заголовков.

Третий параметр принудительно задает код ответа HTTP, хотя можно его указать и в первом параметре. Вот пример отправки заголовка «страница не найдена»:

```
<?php
header('HTTP/1.1 404 Not Found');
```

> ***Внимание!***
>
> Так как функция `header()` устанавливает заголовки ответа сервера, которые посылаются до отсылки основного содержимого документа, то перед функцией не должно быть никаких операторов вывода. Даже пустая строка перед открывающим PHP-дескриптором (`<?php`) вызовет ошибку. Пустые строки внутри PHP-дескрипторов ошибку не генерируют, поскольку вывод информации осуществляется только с помощью операторов вывода. Кроме того, при использовании кодировки UTF-8 файл должен быть сохранен в кодировке UTF-8 без BOM, иначе метка порядка байтов станет причиной ошибки.

> ***Примечание***
>
> Если включена буферизация вывода (см. *разд. 5.1.6*), то перед функцией `header()` могут быть расположены операторы вывода, т. к. данные отправляются не сразу, а только при сбросе буфера.

5.17.4. Перенаправление клиента на другой URL-адрес

Чтобы перенаправить клиента на такой, например, URL: **http://www.rambler.ru/**, нужно написать следующий код:

```
<?php
header("Location: http://www.rambler.ru/");
exit();
```

Предположим, наш сайт содержит четыре страницы: index.php (главная страница), firm.php (о фирме), price.php (продукция) и contact.php (контактная информация). Реализуем механизм навигации по сайту. Переход на другие страницы будет осущест-

вляться не с помощью ссылок, а путем выбора нужной страницы из списка. Для этого на всех страницах сайта должна присутствовать форма, приведенная в листинге 5.130.

Листинг 5.130. Навигация по сайту с помощью списка

```
<form action="go.php">
<select name="page">
<option value="0" selected>На главную</option>
<option value="1">О фирме</option>
<option value="2">Продукция</option>
<option value="3">Контакты</option>
</select>
<input type="submit" value="Go!">
</form>
```

Далее создаем файл go.php с кодом, приведенным в листинге 5.131.

Листинг 5.131. Содержимое файла go.php

```
<?php
if (isset($_GET['page'])) {
   switch($_GET['page']) {
   case '1':
      header('Location: http://localhost/firm.php'); exit();
   case '2':
      header('Location: http://localhost/price.php'); exit();
   case '3':
      header('Location: http://localhost/contact.php'); exit();
   default:
      header('Location: http://localhost/index.php'); exit();
   }
}
else {
   header('Location: http://localhost/index.php'); exit();
}
```

Теперь при выборе страницы из списка и нажатии кнопки **Go!** мы попадем на нужную страницу. Обратите внимание, что вместо оператора `break` мы использовали оператор `exit`, т. к. после перехода на нужную страницу выполнение остального кода просто лишено смысла.

5.17.5. Запрет кэширования страниц

Чтобы запретить кэширование документа, нужно послать сразу несколько заголовков:

```
// Дата в прошлом
header('Expires: Wed, 18 Oct 2017 23:17:32 GMT');
// Дата последнего изменения прямо сейчас
header('Last-Modified: ' . gmdate("D, d M Y H:i:s") . ' GMT');
// Запрет кэширования
header('Cache-Control: no-store, no-cache, must-revalidate');
header('Pragma: no-cache');
```

В заголовке `Expires` можно указать значение `0`, которое также означает «дата в прошлом»:

```
header('Expires: 0'); // Дата в прошлом
```

Если вы думаете, что заголовков слишком много, то ошибаетесь. Иногда и их бывает недостаточно. Например, для браузера Internet Explorer можно дополнительно указать расширения `post-check` и `pre-check` в заголовке `Cache-Control`, хотя в стандарте HTTP 1.1 их нет:

```
header('Cache-Control: no-store, no-cache, must-revalidate');
header('Cache-Control: post-check=0, pre-check=0', false);
```

Если после всех этих «плясок с бубном» оказалось, что контент все равно кэшируется, то следует добавить к URL дополнительный параметр со случайным значением. Это практически всегда помогает!

5.17.6. Реализация ссылки *Скачать*

Очень часто на сайтах можно видеть две ссылки: **Открыть** и **Скачать**. Пусть у нас есть изображение в формате JPEG с именем photo.jpg. Реализовать первую ссылку достаточно просто:

```
<a href="/photo.jpg">Открыть</a>
```

При переходе по ссылке изображение будет открыто без запроса сохранения файла.

Реализовать вторую ссылку позволяет установка соответствующих заголовков. Для этого нам понадобится промежуточный файл, например save.php. В документе размещаем следующую ссылку:

```
<a href="/save.php">Скачать</a>
```

А в файле save.php пишем код, приведенный в листинге 5.132.

Листинг 5.132. Реализация ссылки *Скачать*

```
<?php
$path = 'photo.jpg';
if (!file_exists($path)) {
   echo 'Файл не найден';
}
else {
   $size = filesize($path);
   header('Content-Type: application/octet-stream');
```

```
    header('Content-Length: ' . $size);
    header('Content-Disposition: attachment; filename="' . $path . '"');
    readfile($path);
    exit();
}
```

При переходе по такой ссылке Web-браузер выведет диалоговое окно с запросом «Что делать с файлом?» или сразу сохранит файл в каталог для загруженных файлов (в зависимости от настроек Web-браузера).

> ***Примечание***
>
> В HTML 5 тег `<a>` имеет параметр `download`, который говорит Web-браузеру, что документ нужно скачать, а не открыть (см. *разд. 1.7.1*).

5.17.7. Просмотр заголовков, отправляемых сервером

Посмотреть заголовки, отправляемые удаленным сервером, позволяет функция `get_headers()`. Формат функции:

```
get_headers(<URL-адрес>[, <Разбор заголовков>=0])
```

В параметре `<URL-адрес>` должен быть указан абсолютный путь к файлу:

```
print_r(get_headers('http://localhost/save.php'));
```

По умолчанию функция возвращает список с необработанными заголовками:

```
Array
(
    [0] => HTTP/1.1 200 OK
    [1] => Date: Fri, 05 Jan 2018 22:05:13 GMT
    [2] => Server: Apache/2.4.29 (Win32) OpenSSL/1.1.0g PHP/7.2.0
    [3] => X-Powered-By: PHP/7.2.0
    [4] => Content-Length: 94967
    [5] => Content-Disposition: attachment; filename="photo.jpg"
    [6] => Connection: close
    [7] => Content-Type: application/octet-stream
    [8] => Content-Language: ru
)
```

Если в параметре `<Разбор заголовков>` указать ненулевое значение, то функция вернет ассоциативный массив:

```
print_r(get_headers('http://localhost/save.php', 1));
```

Результат:

```
Array
(
    [0] => HTTP/1.1 200 OK
    [Date] => Fri, 05 Jan 2018 22:07:18 GMT
```

```
    [Server] => Apache/2.4.29 (Win32) OpenSSL/1.1.0g PHP/7.2.0
    [X-Powered-By] => PHP/7.2.0
    [Content-Length] => 94967
    [Content-Disposition] => attachment; filename="photo.jpg"
    [Connection] => close
    [Content-Type] => application/octet-stream
    [Content-Language] => ru
)
```

Получить заголовки, отправленные Web-браузером, внутри программы позволяет функция `apache_request_headers()`:

```
print_r(apache_request_headers());
```

Функция возвращает ассоциативный массив:

```
Array
(
    [Host] => localhost
    [User-Agent] => Mozilla/5.0 (Windows NT 6.2; Win64; x64; rv:56.0)
                    Gecko/20100101 Firefox/56.0
    [Accept] => text/html,application/xhtml+xml,application/xml;
                q=0.9,*/*;q=0.8
    [Accept-Language] => ru-RU,ru;q=0.8,en-US;q=0.5,en;q=0.3
    [Accept-Encoding] => gzip, deflate
    [Connection] => keep-alive
    [Upgrade-Insecure-Requests] => 1
    [Cache-Control] => max-age=0
)
```

Можно также воспользоваться переменной окружения `$_SERVER`:

```
foreach ($_SERVER as $key => $value) {
   if (preg_match('/^HTTP_/isu', $key)) {
      echo "{$key} = {$value}\n";
   }
}
```

Результат:

```
HTTP_HOST = localhost
HTTP_USER_AGENT = Mozilla/5.0 (Windows NT 6.2; Win64; x64; rv:56.0)
                  Gecko/20100101 Firefox/56.0
HTTP_ACCEPT = text/html,application/xhtml+xml,application/xml;
              q=0.9,*/*;q=0.8
HTTP_ACCEPT_LANGUAGE = ru-RU,ru;q=0.8,en-US;q=0.5,en;q=0.3
HTTP_ACCEPT_ENCODING = gzip, deflate
HTTP_CONNECTION = keep-alive
HTTP_UPGRADE_INSECURE_REQUESTS = 1
HTTP_CACHE_CONTROL = max-age=0
```

Как видно из результата, к именам заголовков добавляется префикс `HTTP_` и вместо дефиса вставляется символ подчеркивания.

Получить список заголовков, сформированных внутри программы, позволяют функции `apache_response_headers()` и `headers_list()`:

```
<?php
header('Last-Modified: Thu, 19 Oct 2017 01:19:33 GMT');
print_r(headers_list());
```

Результат:

```
Array
(
    [0] => X-Powered-By: PHP/7.2.0
    [1] => Last-Modified: Thu, 19 Oct 2017 01:19:33 GMT
)
```

5.17.8. Удаление заголовков

Для удаления уже сформированного, но еще не отправленного, заголовка предназначена функция `header_remove()`, а проверить, были ли отправлены заголовки, позволяет функция `headers_sent()`. Давайте удалим заголовок `X-Powered-By`, автоматически сформированный PHP (листинг 5.133).

Листинг 5.133. Удаление заголовка `X-Powered-By`

```
<?php
header('Last-Modified: Thu, 19 Oct 2017 01:19:33 GMT');
if (!headers_sent()) {
   header_remove('X-Powered-By');
}
print_r(headers_list());
```

Результат:

```
Array
(
    [0] => Last-Modified: Thu, 19 Oct 2017 01:19:33 GMT
)
```

Если нужно полностью избавиться от версии PHP в заголовках ответа сервера, то в конфигурационном файле php.ini следует задать директиве `expose_php` значение `Off`:

```
expose_php=Off
```

5.17.9. Работа с cookies

Web-браузеры позволяют сохранять небольшой объем информации на компьютере пользователя. Такая информация называется *cookies*. Помните, что возможность использования cookies можно отключить в настройках Web-браузера.

Для записи cookies предназначена функция `setcookie()`. Формат функции:

```
setcookie(<Имя>, <Значение>[, <Время жизни>[, <Путь>[, <Домен>[,
         <HTTPS>[, <HTTP>]]]]])
```

Большинство параметров здесь необязательные. В третьем параметре указывается количество секунд с начала эпохи. Если не указано `<Время жизни>` cookies, то они будут удалены сразу после закрытия Web-браузера:

```
setcookie('var1', '12');
setcookie('var2', '15', time() + 60 * 60 * 24, '/');
```

Последняя инструкция устанавливает cookies на один день для всех страниц.

> **Внимание!**
>
> Так как функция `setcookie()` устанавливает заголовки ответа сервера, то перед функцией не должно быть никаких операторов вывода. Даже пустая строка перед открывающим PHP-дескриптором (`<?php`) вызовет ошибку. Кроме того, при использовании кодировки UTF-8 файл должен быть в кодировке UTF-8 без BOM.

Считывание cookies осуществляется следующим образом:

```
echo $_COOKIE['var1'] ?? '';
echo $_COOKIE['var2'] ?? '';
```

Все отправленные Web-браузером cookies без обработки доступны через переменную окружения `$_SERVER['HTTP_COOKIE']`:

```
print_r($_SERVER['HTTP_COOKIE']); // var1=12; var2=15
```

Если в директиве `request_order` указана буква `C` (`request_order="GPC"`), то cookies будут также доступны через суперглобальный массив `$_REQUEST`. По умолчанию в пакете XAMPP буква не указана, поэтому cookies через этот массив недоступны. Попробуйте в файл .htaccess добавить следующую директиву:

```
php_value request_order "GPC"
```

После этого через массив `$_REQUEST` можно увидеть все cookies:

```
print_r($_REQUEST);
```

Для удаления cookies следует установить cookies с прошедшей датой, причем параметры должны быть указаны такие же, как и при установке:

```
setcookie('var2', '', time() - 3600, '/');
```

Значение cookies при использовании функции `setcookie()` перед отправкой подвергается URL-кодированию функцией `urlencode()` (см. *разд. 5.7.17*). Если нужно этого избежать, то следует воспользоваться функцией `setrawcookie()`, имеющей точно такой же формат, но не выполняющей URL-кодирование автоматически:

```
setrawcookie('var3', rawurlencode('текст'), time() + 3600, '/');
```

5.17.10. Создаем индивидуальный счетчик просмотров

В качестве примера работы с cookies создадим счетчик просмотров страницы сайта (листинг 5.134).

Листинг 5.134. Счетчик просмотра страницы сайта

```
<?php
$count = $_COOKIE['page_views'] ?? 0;
$count++;
setcookie('page_views', $count, time() + 60 * 60 * 24 * 365);
echo "Вы просмотрели страницу $count раз";
```

5.17.11. Разбор и кодирование URL-адреса

Для разбора URL-адреса на составные части предназначена функция `parse_url()`. Формат функции:

```
parse_url(string $url[, int $component=-1]) : mixed
```

Функция возвращает ассоциативный массив или значение `false` в случае ошибки:

```
$url = 'http://localhost:80/index.php?x=5&y=3#metka';
print_r(parse_url($url));
```

Результат:

```
Array
(
    [scheme] => http
    [host] => localhost
    [port] => 80
    [path] => /index.php
    [query] => x=5&y=3
    [fragment] => metka
)
```

Если нужно получить только конкретный компонент, то во втором параметре нужно указать константы `PHP_URL_SCHEME` (протокол), `PHP_URL_HOST` (домен), `PHP_URL_PORT` (порт), `PHP_URL_PATH` (путь), `PHP_URL_QUERY` (строка запроса), `PHP_URL_FRAGMENT` (якорь), `PHP_URL_USER` (пользователь) или `PHP_URL_PASS` (пароль). В этом случае функция вернет строку, число или значение `null`:

```
$url = 'http://localhost:80/index.php?x=5&y=3#metka';
var_dump(parse_url($url, PHP_URL_SCHEME));   // string(4) "http"
var_dump(parse_url($url, PHP_URL_HOST));     // string(9) "localhost"
var_dump(parse_url($url, PHP_URL_PORT));     // int(80)
var_dump(parse_url($url, PHP_URL_PATH));     // string(10) "/index.php"
```

```
var_dump(parse_url($url, PHP_URL_QUERY));    // string(7) "x=5&y=3"
var_dump(parse_url($url, PHP_URL_FRAGMENT)); // string(5) "metka"
var_dump(parse_url($url, PHP_URL_USER));     // NULL
$url = 'ftp://user:123@localhost:21/index.php';
var_dump(parse_url($url, PHP_URL_USER));     // string(4) "user"
var_dump(parse_url($url, PHP_URL_PASS));     // string(3) "123"
```

Сформировать закодированную строку запроса на основе массива или объекта позволяет функция `http_build_query()`. Формат функции:

```
http_build_query(mixed $query_data[, string $numeric_prefix[,
                 string $arg_separator[, int $enc_type]]]) : string
```

В первом параметре указывается массив или объект. Во втором параметре можно указать префикс, который будет добавляться к числовым индексам списка. Параметр `$arg_separator` позволяет задать разделитель аргументов (по умолчанию используется символ `&`). В параметре `$enc_type` можно указать константы `PHP_QUERY_RFC1738` (символ пробела заменяется плюсом, значение по умолчанию) и `PHP_QUERY_RFC3986` (символ пробела заменяется комбинацией `%20`).

```
$arr = ['x' => 10, 's' => 'кот'];
echo http_build_query($arr); // x=10&s=%D0%BA%D0%BE%D1%82
echo http_build_query($arr, '', '&amp;');
// x=10&amp;s=%D0%BA%D0%BE%D1%82
$arr = ['рыжий кот'];
echo http_build_query($arr, 'var_', '&', PHP_QUERY_RFC1738);
// var_0=%D1%80%D1%8B%D0%B6%D0%B8%D0%B9+%D0%BA%D0%BE%D1%82
echo http_build_query($arr, 'var_', '&', PHP_QUERY_RFC3986);
// var_0=%D1%80%D1%8B%D0%B6%D0%B8%D0%B9%20%D0%BA%D0%BE%D1%82
```

> ***Примечание***
>
> Если нужно выполнить URL-кодирование строки, то следует воспользоваться функциями `urlencode()`, `urldecode()`, `rawurlencode()` и `rawurldecode()` (см. *разд. 5.7.17*).

5.17.12. Получение информации из сети Интернет

При рассмотрении функций для работы с файловой системой в *разд. 5.16* мы оставили без внимания параметр `<Контекст>`. Все функции, имеющие этот параметр, позволяют работать не только с локальной файловой системой, но и с файлами, находящимися на другом сервере в Интернете.

С помощью функций `fopen()`, `file()` и `file_get_contents()` файл можно получить как по протоколу HTTP, так и по протоколу FTP. Для этого достаточно указать соответствующий протокол в URL-адресе, переданном в качестве параметра `<Путь к файлу>` этим функциям, — например, так: **http://www.site.ru/file.txt**.

Для просмотра заголовков и параметров, отправляемых нашим роботом, создадим скрипт testrobots.php (листинг 5.135).

Листинг 5.135. Содержимое файла testrobots.php

```
<?php
print_r(apache_request_headers());
echo "GET: ";
print_r($_GET);
echo "POST: ";
print_r($_POST);
echo "COOKIE: ";
print_r($_COOKIE);
```

Вот пример получения данных по протоколу HTTP методом `GET`:

```
$s = file_get_contents('http://localhost/testrobots.php?x=10');
echo "<pre>$s</pre>";
```

Результат:

```
Array ( [Host] => localhost [Connection] => close )
GET: Array ( [x] => 10 )
POST: Array ()
COOKIE: Array ()
```

Как вам уже известно, в логах сервера отображается программное обеспечение клиента, сделавшего запрос. С помощью директивы `user_agent` в файле php.ini можно задать свое название. Для этого вместо строки:

```
;user_agent="PHP"
```

нужно написать другую, например:

```
user_agent="MySpider/1.0"
```

После этой настройки в первом массиве результата должна появиться запись:

```
[User-Agent] => MySpider/1.0
```

При обработке больших файлов может потребоваться больше времени, чем задано по умолчанию (30 секунд). Увеличить время работы сценария можно с помощью директивы `max_execution_time`:

```
max_execution_time=120
```

Кроме того, следует учитывать, что с помощью директивы `allow_url_fopen` можно запретить открытие внешних файлов. Для получения информации из сети Интернет значение директивы должно быть равно `On`:

```
allow_url_fopen=On
```

Заголовки запроса, а также различные параметры задаются с помощью объекта контекста. Создать этот объект позволяет функция `stream_context_create()`:

```
stream_context_create([array $options[, array $params]]) : resource
```

В первом параметре задаются опции контекста в виде ассоциативного массива, а во втором — ассоциативный массив с параметрами. Для каждого протокола существу-

ет свой набор опций и их очень много. Полный список опций смотрите в документации.

Отправим данные методом `GET` и получим ответ с помощью функций `fopen()` и `fgets()` (листинг 5.136).

Листинг 5.136. Передача данных методом `GET`

```
<?php
$options = [
   'http' => ['method' => 'GET',
              'header' => ['Accept: text/html',
                           'Cookie: test=5'],
              'user_agent' => 'MySpider/1.0',
              'timeout' => 10.0 ]
];
$context = stream_context_create($options);

$query = http_build_query(['x' => 10, 's' => 'кот']);
$host = 'http://localhost/testrobots.php?' . $query;
@$stream = fopen($host, 'rb', false, $context);
if ($stream) {
   echo '<pre>';
   while (($s = fgets($stream, 4096)) !== false) {
      echo $s;
   }
   print_r(stream_context_get_options($stream));
   print_r(stream_context_get_params($stream));
   echo '</pre>';
   fclose($stream);
}
else {
   echo 'Не удалось открыть поток';
}
```

В этом примере мы задали значения следующим опциям:

- `method` — задает метод передачи данных;
- `header` — позволяет добавить заголовки запроса. В качестве значения можно указать массив или строку, внутри которой заголовки разделены символами `\r\n`;
- `user_agent` — задает значение заголовка `User-Agent`;
- `timeout` — ограничивает время ожидания ответа сервера указанным значением.

Получить значения опций потока или контекста позволяет функция `stream_context_get_options()`. Формат функции:

```
stream_context_get_options(resource $stream_or_context) : array
```

С помощью функции `stream_context_get_params()` можно посмотреть значения всех установленных параметров для потока или контекста. Формат функции:

```
stream_context_get_params(resource $stream_or_context) : array
```

Функция `stream_context_set_option()` позволяет установить опции для потока или контекста, а функция `stream_context_set_params()` — задать параметры.

Отправим данные методом `POST` и получим ответ с помощью функции `file_get_contents()` (листинг 5.137).

Листинг 5.137. Передача данных методом `POST`

```
<?php
$query = http_build_query(['x' => 10, 's' => 'кот']);
$options = [
   'http' => ['method' => 'POST',
              'header' => [
                 'Content-Type: application/x-www-form-urlencoded',
                 'Content-Length: ' . strlen($query),
                 'Cookie: test=5'],
              'user_agent' => 'MySpider/1.0',
              'timeout' => 10.0,
              'content' => $query]
];
$context = stream_context_create();
stream_context_set_option($context, $options);

$host = 'http://localhost/testrobots.php';
@$str = file_get_contents($host, false, $context);
if ($str !== false) echo "<pre>$str</pre>";
else echo 'Не удалось получить данные';
```

Обратите внимание: при передаче данных методом `POST` нужно дополнительно указать заголовки `Content-Type` и `Content-Length`:

```
Content-Type: application/x-www-form-urlencoded
Content-Length: длина
```

Строку с данными следует передать в качестве значения опции `content`, а не после знака вопроса в URL-адресе:

```
'content' => $query
```

Если параметр `<Контекст>` не указан явным образом, то используется объект контекста по умолчанию. Получить объект контекста по умолчанию можно с помощью функции `stream_context_get_default()`. Формат функции:

```
stream_context_get_default([array $options]) : resource
```

Пример:

```
$context = stream_context_get_default();
print_r(stream_context_get_options($context));
```

Функция `stream_context_set_default()` позволяет задать опции контекста по умолчанию. Формат функции:

```
stream_context_set_default(array $options) : resource
```

Отправим запрос методом `GET` и получим ответ с помощью функции `file()` (листинг 5.138).

Листинг 5.138. Установка опций контекста по умолчанию

```
<?php
$options_default = [
   'http' => ['method' => 'GET',
              'header' => ['Accept: text/html'],
              'user_agent' => 'MySpider/1.0',
              'timeout' => 10.0 ]
];
stream_context_set_default($options_default);

@$arr = file('http://localhost/testrobots.php');
if ($arr !== false) {
   echo '<pre>' . implode('', $arr) . '</pre>';
}
else echo 'Не удалось получить данные';
```

5.17.13. Функция *fsockopen()*

Функция `fsockopen()` дает возможность получить не только содержимое документа, но и все заголовки ответа сервера. Эта функция очень универсальна и позволяет открыть соединение не только с портом 80, но и с любым другим. Например, можно передать сообщение почтовому серверу на 25-й порт. Формат функции:

```
fsockopen(<Хост>[, <Порт>[, <Номер ошибки>[, <Сообщение об ошибке>[,
         <Тайм-аут>]]]])
```

Функция устанавливает сетевое соединение и возвращает его дескриптор. Если соединение не установлено, то функция возвращает `false`. Получить номер и сообщение об ошибке можно с помощью необязательных параметров `<Номер ошибки>` и `<Сообщение об ошибке>`. В необязательном параметре `<Тайм-аут>` можно указать максимальное время, в течение которого производится попытка соединения (в секундах). Если параметр не указан, то значение будет взято из директивы `default_socket_timeout` файла php.ini:

```
default_socket_timeout=60
```

Обратите внимание: параметр `<Тайм-аут>` задает время ожидания только для соединения. Чтобы задать тайм-аут для операций чтения и записи следует воспользоваться функцией `stream_set_timeout()`. Формат функции:

```
stream_set_timeout(resource $stream, int $seconds[,
                   int $microseconds=0]) : bool
```

После установки соединения необходимо передать заголовки запроса. Между собой заголовки должны разделяться с помощью комбинации символов `\r\n`, а отделяться от тела запроса — с помощью комбинации `\r\n\r\n`.

Открытым соединением можно манипулировать как обычным файлом с помощью функций `fgets()`, `fwrite()`, `feof()` и др. Закрыть соединение позволяет функция `fclose()`.

Функция `stream_set_blocking()` дает возможность установить режим блокировки соединения. Формат функции:

```
stream_set_blocking(<Дескриптор соединения>, <Режим блокировки>)
```

Если режим равен `true`, то функции чтения будут ожидать полного завершения передачи данных. Если указать `false`, то блокировка снимается.

В качестве примера отправим запрос методом `GET` и выведем отдельно заголовки ответа сервера и содержимое ответа (листинг 5.139).

Листинг 5.139. Отправка запроса методом `GET`

```
<?php
$query = http_build_query(['x' => 10, 's' => 'кот']);
$page = '/testrobots.php?' . $query;
$port = '80';
$host = 'localhost';
$header = "GET $page HTTP/1.1\r\n";
$header .= "Host: $host\r\n";
$header .= "User-Agent: MySpider/1.0\r\n";
$header .= "Accept: text/html, text/plain\r\n";
$header .= "Accept-Language: ru, ru-RU\r\n";
$header .= "Accept-Encoding: identity\r\n";
$header .= "Connection: close\r\n";
$header .= "Cookie: test=5\r\n";
$header .= "\r\n";
@$fsock = fsockopen($host, $port, $err, $err_text, 30);
if ($fsock) {
   stream_set_blocking($fsock, true);
   stream_set_timeout($fsock, 5);
   fwrite($fsock, $header);
   $is_header = true;
   $content = '';
   $headers = '';
```

```
   while(($buffer = fgets($fsock, 1024)) !== false) {
      if ($buffer == "\r\n") { $is_header = false; }
      if ($is_header) $headers .= $buffer;
      else            $content .= $buffer;
   }
   fclose($fsock);
}
else {
   echo "Произошла ошибка {$err}: {$err_text}";
}
echo "<b>Заголовки ответа сервера:</b><br><br>\n";
echo "<pre>" . htmlspecialchars($headers) . "</pre>\n";
echo "<br><br><b>Содержимое страницы:</b><br><br>\n";
echo "<pre>" . htmlspecialchars($content) . "</pre>";
```

Обратите внимание на строку:

```
$host = 'localhost';
```

Мы не указали протокол соединения, т. к. протокол `http://` по умолчанию закреплен за 80-м портом.

Если необходимо получить только заголовки ответа сервера, то вместо метода `GET` следует указать метод `HEAD`. Для этого строку:

```
$header = "GET $page HTTP/1.1\r\n";
```

нужно заменить на:

```
$header = "HEAD $page HTTP/1.1\r\n";
```

Данные можно передать не только методами `GET` и `HEAD`, но и методом `POST`, имитируя таким образом передачу данных формы. Рассмотрим это на примере (листинг 5.140).

Листинг 5.140. Передача данных методом POST

```
<?php
$query = http_build_query(['x' => 10, 's' => 'кот']);
$len = strlen($query);
$page = '/testrobots.php';
$port = '80';
$host = 'localhost';
$header = "POST $page HTTP/1.1\r\n";
$header .= "Host: $host\r\n";
$header .= "User-Agent: MySpider/1.0\r\n";
$header .= "Accept: text/html, text/plain\r\n";
$header .= "Accept-Language: ru, ru-RU\r\n";
$header .= "Accept-Encoding: identity\r\n";
$header .= "Connection: close\r\n";
$header .= "Content-Type: application/x-www-form-urlencoded\r\n";
```

```
$header .= "Content-Length: $len\r\n";
$header .= "Cookie: test=5\r\n";
$header .= "\r\n";
@$fsock = fsockopen($host, $port, $err, $err_text, 30);
if ($fsock) {
   stream_set_blocking($fsock, true);
   stream_set_timeout($fsock, 5);
   fwrite($fsock, $header);
   fwrite($fsock, $query);
   $is_header = true;
   $content = '';
   $headers = '';
   while(($buffer = fgets($fsock, 1024)) !== false) {
      if ($buffer == "\r\n") { $is_header = false; }
      if ($is_header) $headers .= $buffer;
      else            $content .= $buffer;
   }
   fclose($fsock);
}
else {
   echo "Произошла ошибка {$err}: {$err_text}";
}
echo "<b>Заголовки ответа сервера:</b><br><br>\n";
echo "<pre>" . htmlspecialchars($headers) . "</pre>";
echo "<br><br><b>Содержимое страницы:</b><br><br>\n";
echo "<pre>" . htmlspecialchars($content) . "</pre>";
```

В этом примере мы заменили метод передачи данных с `GET` на `POST`, отредактировали значение переменной `$page`:

```
$page = '/testrobots.php';
```

и добавили два заголовка:

```
$header .= "Content-Type: application/x-www-form-urlencoded\r\n";
$header .= "Content-Length: $len\r\n";
```

Первый заголовок имитирует передачу данных формы, а второй указывает длину данных, переданных методом `POST`. Сами данные мы передаем после всех заголовков:

```
fwrite($fsock, $query);
```

5.17.14. Использование библиотеки CURL

Вместо рассмотренных функций для получения информации из сети Интернет можно воспользоваться *библиотекой CURL* (Client URL Library). Мы рассмотрим лишь работу с протоколом HTTP, но следует помнить, что библиотека позволяет работать и с другими протоколами, например с FTP. Кроме того, библиотека под-

держивает возможность взаимодействия с прокси-серверами. Полный список ее возможностей смотрите в документации.

Чтобы использовать библиотеку CURL, нужно, чтобы в файле php.ini не было символа комментария (`;`) перед строкой:

```
extension=php_curl.dll
```

В PHP 7.2 строка выглядит так:

```
extension=curl
```

В библиотеку CURL входят следующие основные функции:

- `curl_init([<URL-адрес>])` — создает новый сеанс и возвращает идентификатор;
- `curl_close(<Идентификатор>)` — завершает сеанс;
- `curl_setopt(<Идентификатор>, <Параметр>, <Значение>)` — устанавливает параметры сеанса.

  `<Параметр>` может равняться одной из следующих констант, определяющих смысл параметра `<Значение>` (полный список констант смотрите в документации):

  - `CURLOPT_HTTP_VERSION` — задает версию протокола HTTP. Можно указать константы `CURL_HTTP_VERSION_NONE`, `CURL_HTTP_VERSION_1_0` или `CURL_HTTP_VERSION_1_1`;
  - `CURLOPT_URL` — URL-адрес;
  - `CURLOPT_PORT` — номер порта;
  - `CURLOPT_USERAGENT` — значение HTTP-заголовка `User-Agent`;
  - `CURLOPT_RETURNTRANSFER` — если `true`, то функция `curl_exec()` будет возвращать результат, а не выводить его сразу в Web-браузер;
  - `CURLOPT_TIMEOUT` — максимальное время выполнения операции (в секундах);
  - `CURLOPT_CONNECTTIMEOUT` — максимальное время ожидания установки соединения (в секундах). Для снятия ограничения используется значение `0`;
  - `CURLOPT_HEADER` — если `true`, то результат будет включать не только содержимое документа, но и заголовки ответа сервера;
  - `CURLOPT_NOBODY` — если `true`, то результат будет включать только заголовки ответа сервера. Запрос выполняется методом `HEAD`;
  - `CURLOPT_HTTPGET` — если `true`, то принудительно задает метод `GET` для запроса. Так как метод `GET` устанавливается по умолчанию, использовать этот параметр нужно только в том случае, если ранее был задан другой метод передачи данных;
  - `CURLOPT_POST` — если `true`, то запрос будет отправлен методом `POST`. Кроме того, будет добавлен HTTP-заголовок `Content-Type` со значением `application/x-www-form-urlencoded`. Этот заголовок обычно используется при передаче данных формы;

- `CURLOPT_POSTFIELDS` — строка, содержащая данные для передачи методом `POST`;
- `CURLOPT_REFERER` — значение HTTP-заголовка `Referer`;
- `CURLOPT_COOKIE` — значение HTTP-заголовка `Cookie`;
- `CURLOPT_FOLLOWLOCATION` — если `true`, то перенаправления будут обрабатываться автоматически;
- `CURLOPT_MAXREDIRS` — задает максимальное количество перенаправлений;
- `CURLOPT_HTTPHEADER` — массив с дополнительными HTTP-заголовками;
- `CURLOPT_FILE` — дескриптор файла, в который будет выведен результат операции;
- `CURLOPT_WRITEHEADER` — дескриптор файла, в который будут выведены полученные заголовки;
- `CURLOPT_STDERR` — дескриптор файла, в который будут выводиться сообщения об ошибках;

- `curl_exec(<Идентификатор>)` — выполняет запрос. Если параметр `CURLOPT_RETURNTRANSFER` имеет значение `true`, то функция `curl_exec()` будет возвращать результат, а не выводить его сразу в Web-браузер;
- `curl_getinfo(<Идентификатор>[, <Параметр>])` — возвращает информацию о последней операции. Если `<Параметр>` не указан, то функция возвращает ассоциативный массив. Если `<Параметр>` равен `CURLINFO_HTTP_CODE`, то функция возвращает последний полученный код HTTP. Если `<Параметр>` равен `CURLINFO_CONTENT_TYPE`, то функция возвращает содержимое полученного заголовка `Content-Type` или значение `null`, если заголовка нет в ответе сервера. Полный список констант смотрите в документации;
- `curl_errno(<Идентификатор>)` — возвращает код последней ошибки или `0`, если ошибки не произошло;
- `curl_error(<Идентификатор>)` — позволяет получить строку с описанием последней ошибки.

Пример запроса методом `GET` с получением содержимого документа и всех заголовков ответа сервера приведен в листинге 5.141.

Листинг 5.141. Передача данных методом `GET`

```
<?php
$query = http_build_query(['x' => 10, 's' => 'кот']);
$url = 'http://localhost/testrobots.php?' . $query;
@$curl = curl_init();
if (!curl_errno($curl)) {
   curl_setopt($curl, CURLOPT_URL, $url);
   curl_setopt($curl, CURLOPT_USERAGENT, 'MySpider/1.0');
   curl_setopt($curl, CURLOPT_HEADER, true);
```

```
    $headers = array(
       'Accept: text/html, text/plain',
       'Accept-Language: ru, ru-RU',
       'Accept-Encoding: identity',
       'Connection: close',
       'Cookie: test=5'
    );
    curl_setopt($curl, CURLOPT_HTTPHEADER, $headers);
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
    $content = curl_exec($curl);
    if (!curl_errno($curl)) {
       echo '<b>Код возврата:</b> ';
       echo curl_getinfo($curl, CURLINFO_HTTP_CODE);
       echo '<br><b>Заголовок Content-Type:</b> ';
       echo curl_getinfo($curl, CURLINFO_CONTENT_TYPE);
       echo '<br><br>';
       echo "<b>Содержимое страницы:</b><br><br>\n";
       echo '<pre>' . htmlspecialchars($content) . '</pre>';
    }
    else {
       echo 'Произошла ошибка ' . curl_errno($curl) . ': ';
       echo curl_error($curl);
    }
    curl_close($curl);
}
else {
    echo 'Произошла ошибка ' . curl_errno($curl) . ': ';
    echo curl_error($curl);
}
```

В ряде случаев достаточно получить только заголовки ответа сервера (листинг 5.142).

Листинг 5.142. Получение заголовков ответа сервера

```
<?php
$url = 'http://localhost/testrobots.php';
@$curl = curl_init($url);
if (!curl_errno($curl)) {
    curl_setopt($curl, CURLOPT_USERAGENT, 'MySpider/1.0');
    curl_setopt($curl, CURLOPT_HEADER, true);
    curl_setopt($curl, CURLOPT_NOBODY, true);
    $headers = array(
       'Accept: text/html, text/plain',
       'Accept-Language: ru, ru-RU',
       'Accept-Encoding: identity',
       'Connection: close'
    );
```

```
    curl_setopt($curl, CURLOPT_HTTPHEADER, $headers);
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
    $content = curl_exec($curl);
    if (!curl_errno($curl)) {
       echo "<b>Заголовки ответа сервера:</b><br><br>\n";
       echo '<pre>' . htmlspecialchars($content) . '</pre>';
    }
    else {
       echo 'Произошла ошибка ' . curl_error($curl);
    }
    curl_close($curl);
}
else {
    echo 'Произошла ошибка ' . curl_error($curl);
}
```

Приведем также пример запроса методом `POST`, в котором мы не получаем заголовки ответа сервера, а только выводим содержимое страницы (листинг 5.143).

Листинг 5.143. Передача данных методом `POST`

```
<?php
$url = 'http://localhost/testrobots.php';
@$curl = curl_init($url);
if (!curl_errno($curl)) {
    curl_setopt($curl, CURLOPT_USERAGENT, 'MySpider/1.0');
    curl_setopt($curl, CURLOPT_HEADER, false);
    $headers = array(
       'Accept: text/html, text/plain',
       'Accept-Language: ru, ru-RU',
       'Accept-Encoding: identity',
       'Connection: close',
       'Cookie: test=5'
    );
    curl_setopt($curl, CURLOPT_HTTPHEADER, $headers);
    curl_setopt($curl, CURLOPT_POST, true);
    $query = http_build_query(['x' => 10, 's' => 'кот']);
    curl_setopt($curl, CURLOPT_POSTFIELDS, $query);
    curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
    $content = curl_exec($curl);
    if (!curl_errno($curl)) {
       echo "<b>Содержимое страницы:</b><br><br>\n";
       echo '<pre>' . htmlspecialchars($content) . '</pre>';
    }
    else {
       echo 'Произошла ошибка ' . curl_error($curl);
    }
```

```
   curl_close($curl);
}
else {
   echo 'Произошла ошибка ' . curl_error($curl);
}
```

5.17.15. Отправка писем с сайта

Отправить письма с сайта позволяет функция `mail()`. Формат функции:

```
mail(<E-mail получателя>, <Тема>, <Сообщение>[, <Заголовки>[, <Параметры>]])
```

Функция возвращает `true`, если письмо отправлено.

Чтобы письма отправлялись адресатам, нужно дополнительно настроить программу `sendmail`. На сервере хостинг-провайдера эта программа практически всегда установлена и настроена. Программа `sendmail` входит также в состав пакета XAMPP (C:\xampp\sendmail), но она нуждается в дополнительных настройках и по умолчанию не подключена, поэтому письма отправляться не будут.

Вместо настройки программы `sendmail` мы будем сохранять письма в файл. Для этого в конфигурационном файле php.ini находим строку:

```
;sendmail_path =
```

и заменяем ее на:

```
sendmail_path = "C:/xampp/mailtodisk/mailtodisk.exe"
```

Отправляемые письма будут сохраняться в каталог C:\xampp\mailoutput в виде текстовых файлов.

В параметре `<Заголовки>` обычно указываются следующие заголовки:

- `From` — имя и обратный адрес отправителя:

  ```
  From: Nik <user@mail.ru>
  ```

- `Content-Type` — MIME-тип и кодовая таблица:

  ```
  Content-Type: text/html; charset=windows-1251
  ```

Заголовки должны быть разделены комбинацией символов `\r\n`. Если в тексте заголовков присутствуют русские буквы, то текст следует зашифровать с помощью метода `Base64` следующим образом:

```
=?<Кодировка>?B?<Зашифрованный текст>?=
```

Зашифровать текст с помощью метода `Base64` позволяет функция `base64_encode()`:

```
$subject = 'Сообщение';
$subject = iconv("UTF-8", "windows-1251//IGNORE", $subject);
$subject = "=?windows-1251?B?" . base64_encode($subject) . "?=";
```

Расшифровать зашифрованный ранее текст можно с помощью функции `base64_decode()`:

```
$subject = base64_encode('Сообщение');
echo base64_decode($subject);
```

Зашифровать текст с помощью методов `Base64` или `Quoted-Printable` позволяет функция `mb_encode_mimeheader()` (см. *разд. 5.7.17*). Формат функции:

```
mb_encode_mimeheader(<Строка>[, <Кодировка>[, <Метод кодирования>[,
                     <Символ перевода строк>[, <Отступ>]]]])
```

Пример:

```
mb_internal_encoding('UTF-8');
$subject = 'Сообщение';
echo mb_encode_mimeheader($subject);
// Выведет: =?UTF-8?B?0KHQvtC+0LHRidC10L3QuNC1?=
```

Текст сообщения шифровать необязательно. А вот кодовую таблицу символов следует указать в заголовке `Content-Type`. Также необходимо учитывать, что длина одной строки не должна превышать 70 символов. Строки отделяются друг от друга комбинацией символов: `\r\n`.

Любое письмо может быть отправлено в виде обычного текста (листинг 5.144), а также в формате HTML (листинг 5.145). В первом случае указывается MIME-тип `text/plain`, а во втором — `text/html`.

В качестве примера отправим письмо с подтверждением регистрации.

Листинг 5.144. Пример отправки письма в виде обычного текста

```
<?php
$msg = "Здравствуйте!\r\n\r\n";
$msg .= "Вы успешно зарегистрированы.\r\n\r\n";
$msg .= "http://www.site.ru/\r\n";
$msg .= "support@site.ru";
$msg = iconv("UTF-8", "windows-1251//IGNORE", $msg);
$headers = "Content-Type: text/plain; charset=windows-1251\r\n";
$headers .= "From: Support <support@site.ru>";
$subject = 'Сообщение';
$subject = iconv("UTF-8", "windows-1251//IGNORE", $subject);
$subject = "=?windows-1251?B?" . base64_encode($subject) . "?=";
mail('vasya@mail.ru',  $subject, $msg, $headers);
```

Отправленное письмо (с заголовками) будет выглядеть следующим образом (файл с текстом письма сохраняется в каталоге C:\xampp\mailoutput):

```
To: vasya@mail.ru
Subject: =?windows-1251?B?0e7u4fnl7ejl?=
X-PHP-Originating-Script: 0:index.php
Content-Type: text/plain; charset=windows-1251
From: Support <support@site.ru>
```

```
Здравствуйте!

Вы успешно зарегистрированы.

http://www.site.ru/
support@site.ru
```

Листинг 5.145. Пример отправки письма в формате HTML

```
<?php
mb_internal_encoding('UTF-8');
$msg = "Здравствуйте!<br><br>\n";
$msg .= "Вы успешно зарегистрированы.<br><br>\n";
$msg .= "<a href=\"http://www.site.ru/\">http://www.site.ru/</a><br>\n";
$msg .= "support@site.ru";
$headers = "Content-Type: text/html; charset=UTF-8\r\n";
$headers .= "From: Support <support@site.ru>";
$subject = 'Сообщение';
$subject = mb_encode_mimeheader($subject);
mail('vasya@mail.ru',  $subject, $msg, $headers);
```

5.17.16. Рассылка писем по адресам E-mail из файла

При изучении работы с файлами мы создали файл file.txt со списком рассылки и механизм работы с ним (см. *разд. 5.16.5*). Теперь рассмотрим возможность рассылки писем по адресам E-mail из этого файла (листинг 5.146).

Листинг 5.146. Рассылка писем

```
<form action="<?=$_SERVER['SCRIPT_NAME']?>">
<input type="hidden" name="send" value="1">
<input type="submit" value="Разослать">
</form>
<?php
if (isset($_GET["send"])) {
   require_once 'mail_script.inc';    // Листинг 5.123
   try {
      $arr = read_file_to_array('file.txt');
   } catch (Exception $e) {
      exit($e->getMessage());
   }
   $msg = "Добрый день!\r\n\r\n";
   $msg .= "Новости нашего сайта.\r\n\r\n";
   $msg .= "http://www.site.ru/\r\n";
   $msg .= "mail@site.ru";
   $msg = iconv("UTF-8", "windows-1251//IGNORE", $msg);
   $headers = "Content-Type: text/plain; charset=windows-1251\r\n";
```

```
   $headers .= "From: News <mail@site.ru>";
   $subject = 'Новости сайта';
   $subject = iconv("UTF-8", "windows-1251//IGNORE", $subject);
   $subject = "=?windows-1251?B?" . base64_encode($subject) . "?=";
   foreach ($arr as $email) {
      if (test_email($email)) {
         @mail($email, $subject, $msg, $headers);
      }
   }
   echo 'Сообщения разосланы';
}
```

При нажатии кнопки **Разослать** на все адреса E-mail из файла будет отправлено письмо.

> **Внимание!**
>
> С локальной машины письма отправлены не будут, т. к. мы не настраивали программу отправки писем sendmail (*C:\xampp\sendmail*). На сервере хостинг-провайдера эта программа практически всегда установлена и настроена. Правда, число одновременно отправленных писем часто ограничено.

При рассылке в письме обязательно должна быть предусмотрена возможность отписаться от рассылки. И запомните — рассылка спама в Интернете запрещена. Под спамом понимаются письма, не запрошенные получателем явным образом.

5.18. Обработка данных формы

Рассмотрим способы обработки данных каждого элемента формы по отдельности. Напомним, что о работе с полями для выбора файла мы уже говорили в *разд. 5.16.9*.

5.18.1. Текстовое поле, поле ввода пароля и скрытое поле

После отправки формы, содержащей поля:

```
<input type="text" name="txt">
<input type="password" name="passw">
<input type="hidden" name="hid" value="5">
```

в зависимости от метода передачи данных на сервере будут созданы следующие переменные окружения:

- метод GET:

  ```
  $_GET["txt"]
  $_GET["passw"]
  $_GET["hid"]
  ```

- метод POST:

  ```
  $_POST["txt"]
  $_POST["passw"]
  $_POST["hid"]
  ```

Название массива совпадает с методом передачи данных, а имя ключа — со значением параметра name тега <input>. Все переданные данные, вне зависимости от используемого метода, доступны через массив $_REQUEST.

Предположим, что в эти поля в сценарии необходимо ввести первоначальные значения. Здесь нас может подстерегать проблема. Если вывести так:

```
$str = 'Группа "Кино"';
echo '<input type="text" name="txt" value="' . $str . '">';
```

то в результате поле будет содержать текст "Группа ", а не "Группа "Кино"".

Помните, что при подстановке значений все специальные символы в строке следует заменить HTML-эквивалентами. Кроме того, значение должно быть расположено внутри кавычек. Правильно будет так:

```
$str = 'Группа "Кино"';
$str = htmlspecialchars($str, ENT_COMPAT | ENT_HTML5, 'UTF-8');
echo '<input type="text" name="txt" value="' . $str . '">';
```

HTML-код будет выглядеть следующим образом:

```
<input type="text" name="txt" value="Группа &quot;Кино&quot;">
```

5.18.2. Поле для ввода многострочного текста

После отправки формы:

```
<textarea name="txt">Текст</textarea>
```

в зависимости от метода передачи данных на сервере будут созданы следующие переменные окружения:

- метод GET:

  ```
  $_GET["txt"]
  ```

- метод POST:

  ```
  $_POST["txt"]
  ```

Предположим, что в поле ввода многострочного текста необходимо ввести в сценарии первоначальное значение. Это можно сделать так:

```
$str = 'Привет всем';
echo '<textarea name="txt" cols="15" rows="10">' . $str . '</textarea>';
```

Однако если строка содержит теги, то могут возникнуть проблемы. Поэтому все специальные символы в строке следует заменить HTML-эквивалентами:

```
$str = 'Привет </textarea> всем';
$str = htmlspecialchars($str, ENT_COMPAT | ENT_HTML5, 'UTF-8');
echo '<textarea name="txt" cols="15" rows="10">' . $str . '</textarea>';
```

5.18.3. Список с возможными значениями

После отправки формы, содержащей список:

```
<select name="color">
<option value="1">White</option>
<option>Red</option>
</select>
```

в зависимости от метода передачи данных на сервере будут созданы следующие переменные окружения:

- метод `GET`:

  ```
  $_GET["color"]
  ```

- метод `POST`:

  ```
  $_POST["color"]
  ```

Название массива совпадает с методом передачи данных, а имя ключа — со значением параметра `name` тега `<select>`. Все переданные данные вне зависимости от используемого метода доступны через массив `$_REQUEST`.

Значение переменной будет присвоено в зависимости от пункта, выбранного в списке. Если выбран пункт **White**, то переменная `$_GET["color"]` будет иметь значение `1` (значение параметра `value`). Если выбран пункт **Red**, то переменная `$_GET["color"]` будет иметь значение `"Red"`, т. к. параметр `value` отсутствует.

Если в списке можно выбрать сразу несколько значений, то в параметре `name` после имени следует указать квадратные скобки (символ массива). Все выбранные значения будут помещены в массив. Рассмотрим это на примере (листинг 5.147).

Листинг 5.147. Выбор нескольких значений из списка

```
<form action="<?=$_SERVER['SCRIPT_NAME']?>">
<select name="days[]" size="7" multiple>
<option value="1">Понедельник</option>
<option value="2">Вторник</option>
<option value="3">Среда</option>
<option value="4">Четверг</option>
<option value="5">Пятница</option>
<option value="6">Суббота</option>
<option value="7">Воскресенье</option>
</select><br>
<input type="submit" value="Отправить">
</form>
<?php
if (isset($_GET['days']) && is_array($_GET['days'])) {
   echo "Выбранные пункты<br>\n";
   foreach ($_GET['days'] as $item) {
      echo "{$item}<br>\n";
   }
}
```

5.18.4. Флажок

После отправки формы:

```
<input type="checkbox" name="check1" value="1"> Текст
<input type="checkbox" name="check2"> Текст
```

в случае, если флажки установлены, на сервере будут созданы следующие переменные окружения:

- метод GET:

  ```
  $_GET["check1"]
  $_GET["check2"]
  ```

- метод POST:

  ```
  $_POST["check1"]
  $_POST["check2"]
  ```

Если флажки установлены, то переменные будут иметь следующие значения: переменная `$_GET["check1"]` — `1` (значение параметра `value`), а переменная `$_GET["check2"]` — `on` (нет параметра `value`).

Если флажки не установлены, то переменные не создаются! Поэтому необходимо проверять существование переменной:

```
if (isset($_GET['check1'])) echo $_GET['check1'] . '<br>';
if (isset($_GET['check2'])) echo $_GET['check2'] . '<br>';
```

Если флажки объединены в группу, то после имени следует указать квадратные скобки. Значение параметра `name` у всех флажков должно быть одинаковым, а значение параметра `value` — разным (листинг 5.148).

Листинг 5.148. Объединение флажков в группу

```
<form action="<?=$_SERVER['SCRIPT_NAME']?>">
<input type="checkbox" name="check[]" value="1"> Текст1
<input type="checkbox" name="check[]" value="2"> Текст2
<input type="checkbox" name="check[]" value="3"> Текст3
<input type="submit" value="Отправить">
</form>
<?php
if (isset($_GET['check']) && is_array($_GET['check'])) {
   echo 'Выбранные флажки<br>';
   foreach ($_GET['check'] as $item) {
      echo $item . '<br>';
   }
}
```

5.18.5. Элемент-переключатель

После отправки формы, содержащей такой код HTML:

```
<input type="radio" name="sex" value="1" checked> Мужской
<input type="radio" name="sex" value="2"> Женский
```

в зависимости от метода передачи данных на сервере будут созданы следующие переменные окружения:

- метод GET:

```
$_GET["sex"]
```

- метод POST:

```
$_POST["sex"]
```

Значение переменной `$_GET["sex"]` зависит от выбранного переключателя (`1` или `2`). Если ни один из переключателей не выбран, то переменные не создаются!

5.18.6. Кнопка *Submit*

После отправки формы:

```
<input type="submit" name="go" value="Отправить">
```

в зависимости от метода передачи данных на сервере будут созданы следующие переменные окружения:

- метод GET:

```
$_GET["go"]
```

- метод POST:

```
$_POST["go"]
```

Зачем для кнопки указывать параметр `name`? Все дело в том, что в одной форме может быть несколько кнопок **Submit**. Кроме того, если наш сценарий обрабатывает сразу несколько форм, то это позволит определить, какая форма отправлена. Также часто один и тот же сценарий и отображает форму, и обрабатывает ее данные. Если форма отправлена, то переменная будет существовать, если не отправлена, то переменная создана не будет:

```
if (isset($_GET['go'])) {
   echo 'Форма отправлена';
}
else {
   // Вывести форму
}
```

Для этой же цели может использоваться скрытое поле:

```
<form action="<?=$_SERVER['SCRIPT_NAME']?>">
<input type="text" name="txt">
```

```
<input type="hidden" name="go" value="send">
<input type="submit" value="Отправить">
</form>
```

5.18.7. Проверка корректности данных. Создание формы регистрации пользователя

Рассмотрим форму регистрации пользователя с проверкой корректности введенных данных, а заодно и обработку данных в кодировке UTF-8 (листинг 5.149).

Листинг 5.149. Проверка корректности данных

```
<?php
// Файл в кодировке UTF-8 без BOM !!!
mb_internal_encoding('UTF-8'); // Установка кодировки
if (isset($_POST['go'])) {     // Если форма отправлена
   // Создаем короткие имена
   $user   = $_POST['user']   ?? '';
   $fam    = $_POST['fam']    ?? '';
   $age    = $_POST['age']    ?? '0';
   $email  = $_POST['email']  ?? '';
   $passw1 = $_POST['passw1'] ?? '';
   $passw2 = $_POST['passw2'] ?? '';
   // Создаем переменную для описания всех ошибок
   $err = array();
   // Проверяем корректность введенных данных
   if (mb_strlen($user) > 50 || mb_strlen($user) < 2) {
      $err[] = 'Недопустимая длина поля Имя';
   }
   else {
      if (!preg_match('/^[a-zа-яё -]+$/isu', $user)) {
         $err[] = 'Недопустимые символы в поле Имя';
      }
   }
   if (mb_strlen($fam) > 50 || mb_strlen($fam) < 2) {
      $err[] = 'Недопустимая длина поля Фамилия';
   }
   else {
      if (!preg_match('/^[a-zа-яё -]+$/isu', $fam)) {
         $err[] = 'Недопустимые символы в поле Фамилия';
      }
   }
   $age = intval($age, 10);
   if ($age < 3 || $age > 120) {
      $err[] = 'Неверный возраст';
   }
```

```
   $pattern = '/^([a-z0-9_.-]+)@([a-z0-9-]+\\.)+[a-z]{2,6}$/isu';
   if (mb_strlen($email) > 50 || !preg_match($pattern, $email)) {
      $err[] = 'Неверный адрес E-mail';
   }
   if (!preg_match('/^[a-z0-9_.-]{6,16}$/isu', $passw1)) {
      $err[] = 'Неверный пароль';
   }
   else {
      if ($passw1 !== $passw2) {
         $err[] = 'Пароли должны совпадать';
      }
   }
   // Если ошибок нет, то массив $err будет пустой
   if (count($err) == 0) {
      // Добавляем данные в базу данных
      // и отправляем подтверждение на E-mail
      // Делаем автоопредирект, чтобы очистить данные формы
      header('Location: http://localhost' .
             $_SERVER['SCRIPT_NAME'] . '?reg=ok');
      exit(); // Завершаем работу скрипта
   }
   else { // Если возникли ошибки
    // Заменяем все спецсимволы на HTML-эквиваленты
    $user = htmlspecialchars($user, ENT_COMPAT | ENT_HTML5, 'UTF-8');
    $fam = htmlspecialchars($fam, ENT_COMPAT | ENT_HTML5, 'UTF-8');
    $age = htmlspecialchars($age, ENT_COMPAT | ENT_HTML5, 'UTF-8');
    $email = htmlspecialchars($email, ENT_COMPAT | ENT_HTML5, 'UTF-8');
   }
}
else { // Если форма не отправлена
   $user = $fam = $age = $email = '';
}
header('Content-Type: text/html; charset=utf-8');
?>
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Регистрация пользователя</title>
</head>
<body>
<h2>Регистрация пользователя</h2>
<?php
if (isset($err) && count($err) > 0) {
   echo '<div style="color:red">При заполнении формы были';
   echo ' допущены ошибки:<br><br>';
   echo implode("<br>\n", $err), "<br><br></div>\n";
}
```

```
if (isset($_GET['reg']) && $_GET['reg'] == 'ok') {
   // Если регистрация прошла успешно, выводим подтверждение
   echo '<div style="color:green">';
   echo '<b>Регистрация прошла успешно</b><br><br></div>';
}
?>
<form action="<?=$_SERVER['SCRIPT_NAME']?>" method="POST">
<div>
Имя:<br>
<input type="text" name="user" value="<?=$user ?? ''?>"><br>
Фамилия:<br>
<input type="text" name="fam" value="<?=$fam ?? ''?>"><br>
Возраст:<br>
<input type="text" name="age" value="<?=$age ?? ''?>"><br>
E-mail:<br>
<input type="text" name="email" value="<?=$email ?? ''?>"><br>
Пароль:<br>
<input type="password" name="passw1"><br>
Повторите пароль:<br>
<input type="password" name="passw2">
<br><br>
<input type="submit" name="go" value="Отправить">
</div>
</form>
</body>
</html>
```

Как видно из примера, все имена полей, заданные с помощью параметра `name`, доступны через переменную окружения `$_POST`. Если форма отправлена (переменная `$_POST['go']` будет существовать), то создаем короткие имена переменных. Может возникнуть вопрос, почему просто не присвоить значения напрямую:

```
$user = $_POST['user'];
```

Все дело в том, что если переменная `$_POST['user']` не будет определена, то при включенном режиме вывода всех ошибок интерпретатор выдаст предупреждающее сообщение:

```
Notice: Undefined index: user
```

Избежать появления таких сообщений можно, выключив вывод предупреждающих сообщений, например, с помощью функции `error_reporting()`:

```
error_reporting(E_ALL & ~E_NOTICE);
```

Однако лучше предварительно проверить существование переменной и присвоить ей значение по умолчанию, если она не существует:

```
$user = $_POST['user'] ?? '';
```

Чтобы сохранить описания всех ошибок, сделанных пользователем при вводе, мы определяем переменную `$err` и присваиваем ей пустой массив. Затем проверяем

корректность введенных данных. С помощью функции `mb_strlen()` контролируем длину строк, а с помощью регулярных выражений и функции `preg_match()` проверяем соответствие E-mail шаблону. Если во время проверки возникли ошибки, то добавляем их в массив `$err`.

Если ошибок нет (массив `$err` останется пустым), добавляем данные в базу и отсылаем подтверждение на E-mail.

Далее делаем автоматический редирект и завершаем работу скрипта:

```
header('Location: http://localhost' .
       $_SERVER['SCRIPT_NAME'] . '?reg=ok');
exit();
```

Зачем нужен авторедирект? После отправки данных они сохраняются в кэше Web-браузера, и если нажать кнопку **Обновить** на панели инструментов Web-браузера, то данные будут повторно отправлены серверу. А после авторедиректа нажатие кнопки **Обновить** не будет приводить к повторной отправке данных формы.

Если при проверке были выявлены ошибки, то необходимо заново отобразить форму, вывести сообщения об ошибках и заполнить все поля для редактирования. Так как данные могут содержать специальные символы, мы с помощью функции `htmlspecialchars()` заменяем их на HTML-эквиваленты. После этого можно без опаски заполнить все поля, подставив значения в параметр `value`.

Если форма не была отправлена, то присваиваем переменным пустую строку и выводим форму. В этом случае параметр `value` в элементах формы будет равен пустой строке.

Чтобы документ был правильно обработан Web-браузером, желательно с помощью функции `header()` указать MIME-тип и кодировку документа:

```
header('Content-Type: text/html; charset=utf-8');
```

Часто на хостингах встречается ситуация, когда сервер настроен на кодировку windows-1251. В этом случае сервер автоматически отправит заголовок:

```
Content-Type: text/html; charset=windows-1251
```

Учитывая, что мы работаем с кодировкой UTF-8, Web-браузер может неправильно распознать кодировку, и русские буквы будут искажены. По этой причине рекомендуем всегда явно указывать кодировку, посылая соответствующий заголовок с помощью функции `header()`.

Еще раз напомним: поскольку функция `header()` устанавливает заголовки ответа сервера, которые посылаются до отсылки основного содержимого документа, то перед функцией не должно быть никаких операторов вывода. Даже пустая строка перед открывающим PHP-дескриптором (`<?php`) вызовет ошибку. Пустые строки внутри PHP-дескрипторов ошибку не генерируют, поскольку вывод информации осуществляется только с помощью операторов вывода. Кроме того, при использовании кодировки UTF-8 файл должен быть сохранен в кодировке UTF-8 без BOM, иначе метка порядка байтов станет причиной ошибки.

5.19. Аутентификация с помощью PHP

В *разд 4.4.14* мы уже рассматривали аутентификацию посетителей при помощи файла конфигурации сервера Apache .htaccess. В PHP существует свой способ аутентификации посетителей на основе механизма сессий.

5.19.1. Директивы для управления механизмом сессий

За механизм сессий в файле php.ini отвечают следующие основные директивы:

- `session.save_handler` — определяет способ хранения данных сеанса:

```
session.save_handler=files
```

- `session.save_path` — задает путь к месту хранения данных сеанса:

```
session.save_path="C:/xampp/tmp"
```

- `session.use_cookies` — включает возможность использования cookies для механизма сессий. Если указано значение `1`, то использование разрешено, если `0` — то запрещено:

```
session.use_cookies=1
```

- `session.name` — определяет имя сеанса:

```
session.name=PHPSESSID
```

- `session.auto_start` — задает возможность автоматического запуска механизма работы с сеансами. Значение `0` запрещает запуск:

```
session.auto_start=0
```

- `session.cookie_path` — определяет путь для установки в cookies сеанса:

```
session.cookie_path=/
```

- `session.cache_expire` — устанавливает время жизни для кэшированных страниц сеанса в минутах:

```
session.cache_expire=180
```

- `session.cookie_lifetime` — определяет время жизни cookies на машине пользователя. Значение `0` указывает на то, что cookies будет удалено сразу после закрытия окна Web-браузера:

```
session.cookie_lifetime=0
```

- `session.use_trans_sid` — задает возможность присоединения `PHPSESSID` к URL-адресам. Если указано значение `1`, то использование разрешено, если `0` — то запрещено:

```
session.use_trans_sid=0
```

- `session.use_only_cookies` — если указано значение `1`, то идентификатор сессии будет сохраняться только в cookies:

```
session.use_only_cookies=1
```

5.19.2. Функции для управления сессиями

Работа с механизмом сессий осуществляется следующим образом:

1. Запускается сеанс.
2. Регистрируются переменные сеанса.
3. Используются переменные сеанса.
4. Удаляются переменные сеанса.
5. Сеанс завершается.

Запустить сессию позволяет функция `session_start([<Опции>])`:

```
session_start();
```

Если запустить сессию удалось, функция возвращает `true`.

При запуске сессии на компьютер пользователя устанавливается cookies с именем `PHPSESSID` и значением вида `rbgvdgk9g7sk9hee4cqndm0561`. А в каталоге C:\xampp\tmp создается временный файл сессии с именем `sess_rbgvdgk9g7sk9hee4cqndm0561`. Дополнительно отправляются заголовки, запрещающие кэширование страницы.

Если прием cookies отключен в настройках Web-браузера, то (при условии, что директива `session.use_trans_sid` имеет значение `1` и директива `session.use_only_cookies` имеет значение `0`) к любой ссылке будет добавлена следующая строка:

```
?PHPSESSID=rbgvdgk9g7sk9hee4cqndm0561
```

Если на странице имеется форма, то внутрь будет автоматически добавлено скрытое поле:

```
<input type="hidden" name="PHPSESSID"
       value="rbgvdgk9g7sk9hee4cqndm0561" />
```

При первом запуске будут установлены cookies, а имя сессии будет добавлено в URL-адреса для всех внутренних ссылок. Если cookies разрешено использовать, то в дальнейшем добавление к URL будет прекращено.

Включить автоматическое добавление идентификатора сессии к URL-адресам можно в конфигурационном файле php.ini или внутри программы с помощью параметра `<Опции>` функции `session_start()`:

```
session_start(['use_trans_sid' => 1, 'use_only_cookies' => 0]);
```

> ***Внимание!***
>
> Названия директив указываются без префикса `session.`

Идентификатор сессии в этом случае будет доступен через константу `SID`:

```
echo SID; // PHPSESSID=rbgvdgk9g7sk9hee4cqndm0561
```

Если удалось записать идентификатор сессии в cookies, то константа `SID` будет содержать пустую строку.

Получить идентификатор сессии можно с помощью функции `session_id()`:

```
echo session_id(); // rbgvdgk9g7sk9hee4cqndm0561
```

Следует учитывать, что идентификатор сессии, прикрепленный к URL-адресу, пользователи могут опубликовать на других страницах в Интернете, что приведет к перехвату сессии другим пользователем. Поэтому лучше использовать только cookies или предусматривать дополнительную защиту от перехвата сессии, например, проверяя сигнатуру Web-браузера или IP-адрес пользователя.

Поменять имя сеанса можно с помощью функции `session_name()`:

```
session_name("Ivan");
session_start();
```

Теперь вместо имени `PHPSESSID` будет использоваться имя `Ivan`. Если параметр не указан, то функция вернет используемое имя, в противном случае — старое имя.

Зарегистрировать переменную внутри сеанса можно следующим образом:

```
$_SESSION["var1"] = 1;
```

Переменная сохраняется не в cookies пользователя, а в специальном файле на сервере. В cookies пользователя сохраняется только переменная с именем `PHPSESSID` и значением вида `rbgvdgk9g7sk9hee4cqndm0561`.

Проверить существование переменной можно с помощью оператора `isset()`:

```
if (isset($_SESSION["var1"])) {
   echo "Переменная зарегистрирована";
}
```

Удалить переменную сеанса позволяет оператор `unset()`:

```
unset($_SESSION["var1"]);
```

Удалить сразу все переменные сеанса позволяет функция `session_unset()`:

```
session_start();   // Запускаем сессию
session_unset();   // Удаляем все переменные
session_destroy(); // Удаляем идентификатор
```

Завершая сеанс, сначала нужно удалить все переменные сеанса, а затем вызвать функцию `session_destroy()` для удаления идентификатора.

После запуска сессии файл, в который записываются переменные сессии, блокируется и становится недоступен до завершения работы скрипта. Если выполняется много запросов, то это может стать проблемой. Чтобы избежать блокировки на длительное время, нужно воспользоваться следующими способами:

- вызвать функцию `session_write_close()` после окончания изменения переменных сеанса. Помните, что после вызова этой функции измененные значения переменных сеанса не будут записаны в файл;
- если мы только читаем данные из файла и не изменяем значения переменных сеанса, то в параметре `<Опции>` функции `session_start()` можно указать опцию `read_and_close`:

  ```
  session_start(['read_and_close' => true]);
  ```

5.19.3. Создание Личного кабинета

Для примера создадим в каталоге C:\xampp\htdocs каталог cabinet и в нем разместим следующие файлы:

- index.php — содержит форму для ввода логина и пароля (листинг 5.150);
- cabinet.php — файл с информацией только для прошедших аутентификацию пользователей (листинг 5.151);
- exit.php — для завершения сеанса (листинг 5.152).

Кроме того, в каталоге C:\xampp\php\includes разместим файл data.inc — для хранения логина и пароля (листинг 5.153).

Листинг 5.150. Содержимое файла C:\xampp\htdocs\cabinet\index.php

```
<?php
require_once('data.inc');
$err = '';
if (isset($_POST['login']) && isset($_POST['passw'])) {
   $_POST['passw'] = md5($_POST['passw']);
   if ($_POST['login'] === $enter_login &&
       $_POST['passw'] === $enter_passw) {
      session_start();
      $_SESSION['sess_login'] = $_POST['login'];
      $_SESSION['sess_passw'] = $_POST['passw'];
      header('Location: http://localhost/cabinet/cabinet.php');
      exit();
   }
   else {
      $err = '<div style="color: red"><b>';
      $err .= 'Логин или пароль введены неправильно!';
      $err .= '</b></div>';
   }
}
?>
<h1>Вход в систему</h1>
<form action="index.php" method="POST">
<div>
<b>Логин:</b><br>
<input type="text" name="login"><br>
<b>Пароль:</b><br>
<input type="password" name="passw"><br>
<input type="submit" value="Войти">
</div>
</form>
<?php echo $err; ?>
```

Листинг 5.151. Содержимое файла C:\xampp\htdocs\cabinet\cabinet.php

```
<?php
session_start();
require_once('data.inc');
if (isset($_SESSION['sess_login']) && isset($_SESSION['sess_passw'])) {
   if ($_SESSION['sess_login'] === $enter_login &&
       $_SESSION['sess_passw'] === $enter_passw) {
      echo "Информация для прошедших аутентификацию<br><br>\n";
      echo 'Имя сессии: ' . session_name() . "<br>\n";
      echo 'Идентификатор сессии: ' . session_id() . "<br>\n";
      echo "<a href=\"exit.php\">Выйти из системы</a>\n";
   }
   else {
      header('Location: http://localhost/cabinet/');
      exit();
   }
}
else {
   header('Location: http://localhost/cabinet/');
   exit();
}
```

Листинг 5.152. Содержимое файла C:\xampp\htdocs\cabinet\exit.php

```
<?php
session_start();
session_unset();   // Удаляем все переменные
setcookie(session_name(), '', 0, '/');
session_destroy(); // Удаляем идентификатор
header("Location: http://localhost/cabinet/");
exit();
```

Листинг 5.153. Содержимое файла C:\xampp\php\includes\data.inc

```
<?php
$enter_login = "login";
$enter_passw = "202cb962ac59075b964b07152d234b70";
```

В нашем примере только один логин (`login`) и один пароль (`123`). В реальной практике для каждого пользователя создаются свои логин и пароль. Учетные записи хранятся в файле или, чаще всего, в базе данных. Если используется обычный файл (как в нашем случае), то он должен содержать пароль в зашифрованном виде, а сам файл должен быть недоступен через Интернет. Поэтому в рассматриваемом примере файл data.inc должен быть расположен в каталоге C:\xampp\php\includes, а не в C:\xampp\htdocs\cabinet.

Внимание!

Так как мы устанавливаем заголовки ответа сервера, то перед функцией `session_start()` не должно быть никаких операторов вывода. Даже пустая строка перед открывающим PHP-дескриптором (`<?php`) вызовет ошибку. Кроме того, при использовании кодировки UTF-8 файл должен быть сохранен в кодировке UTF-8 без BOM, иначе метка порядка байтов станет причиной ошибки.

5.20. Работа с графикой

Для работы с графикой предназначена *библиотека GD*. Чтобы использовать библиотеку GD, нужно, чтобы в файле php.ini не было символа комментария (`;`) перед строкой:

```
extension=php_gd2.dll
```

В PHP 7.2 строка выглядит следующим образом:

```
extension=gd2
```

5.20.1. Получение информации о библиотеке GD

Получить информацию о библиотеке GD в виде ассоциативного массива позволяет функция `gd_info()`:

```
<?php
$info = gd_info();
foreach ($info as $key => $value) {
   if ($value === true) {
      $value = 'Да';
   }
   elseif ($value === false) {
      $value = 'Нет';
   }
   echo "{$key}: {$value}<br>\n";
}
```

Результат в окне Web-браузера:

```
GD Version: bundled (2.1.0 compatible)
FreeType Support: Да
FreeType Linkage: with freetype
GIF Read Support: Да
GIF Create Support: Да
JPEG Support: Да
PNG Support: Да
WBMP Support: Да
XPM Support: Да
XBM Support: Да
WebP Support: Да
BMP Support: Да
JIS-mapped Japanese Font Support: Нет
```

Как видно из результата, библиотека GD позволяет работать со следующими основными форматами изображений:

- JPEG (Joint Photographic Experts Group) — этот формат подходит для вставки на Web-страницу фотографий, но из-за артефактов сжатия не подходит для работы с графикой. Прозрачность и анимация не поддерживаются;
- GIF (Graphics Interchange Format) — палитра ограничена 256 цветами. Поддерживает прозрачность и анимацию. Отлично подходит для создания рекламных баннеров;
- PNG (Portable Network Graphics) — поддерживает палитровые и полноцветные изображения, а также прозрачность. При сохранении использует сжатие без потерь, поэтому отлично подходит для работы с графикой, но фотографии, сохраненные в этом формате, имеют большой размер файла. Следует учитывать, что некоторые Web-браузеры могут не поддерживать прозрачность при использовании этого формата.

5.20.2. Загрузка изображения из файла

Для загрузки изображения из файла предусмотрены следующие функции:

- `imagecreatefromjpeg(<Имя файла>)` — для изображений в формате JPEG;
- `imagecreatefromgif(<Имя файла>)` — для изображений в формате GIF;
- `imagecreatefrompng(<Имя файла>)` — для изображений в формате PNG.

Если изображение успешно загружено, то функции возвращают идентификатор ресурса, в противном случае — значение `false`:

```
@$img = imagecreatefromjpeg('C:\\xampp\\htdocs\\photo.jpg');
if ( is_resource($img) ) {
   echo 'Изображение успешно загружено';
   echo gettype($img); // resource
}
```

5.20.3. Создание нового изображения

Функция `imagecreatetruecolor()` позволяет создать новое полноцветное изображение. Если операция успешно выполнена, то функция вернет идентификатор ресурса, в противном случае — значение `false`. Формат функции:

```
<Идентификатор> = imagecreatetruecolor(<Ширина>, <Высота>);
```

Пример:

```
@$img = imagecreatetruecolor(468, 60);
if ( is_resource($img) ) {
   echo 'Изображение успешно создано';
}
```

Создать изображение с палитрой цветов можно с помощью функции `imagecreate()`:

```
<Идентификатор> = imagecreate(<Ширина>, <Высота>);
```

Вместо этой функции для создания изображения рекомендуется использовать функцию `imagecreatetruecolor()`, а затем в случае необходимости выполнять преобразование полноцветного изображения в палитровое с помощью функции `imagetruecolortopalette()`:

```
imagetruecolortopalette(resource $image, bool $dither, int $ncolors) : bool
```

Параметр `$dither` указывает, нужно ли использовать сглаживание, а параметр `$ncolors` задает максимальное число цветов в палитре.

Проверить тип изображения позволяет функция `imageistruecolor()`. Функция возвращает значение `true`, если изображение является полноцветным, и `false` — в противном случае:

```
@$img = imagecreatetruecolor(468, 60);
if ( is_resource($img) ) {
   var_dump(imageistruecolor($img));          // bool(true)
   imagetruecolortopalette($img, false, 255);
   var_dump(imageistruecolor($img));          // bool(false)
}
```

Динамическое создание изображений не имело бы смысла при отсутствии возможности формировать их содержимое. Библиотека GD предоставляет множество функций для изменения созданных или загруженных изображений, которые мы рассмотрим в следующих разделах.

5.20.4. Вывод изображения в Web-браузер

Чтобы вывести изображение в Web-браузер, нужно вначале сформировать соответствующий заголовок с помощью функции `header()`:

```
header('Content-Type: image/jpeg');
header('Content-Type: image/gif');
header('Content-Type: image/png');
```

а затем вывести изображение с помощью соответствующей формату функции:

- `imagejpeg()` — для изображений в формате JPEG:

  ```
  imagejpeg(<Идентификатор>[, <Имя файла>[, <Сжатие>]]) : bool
  ```

  В параметре `<Сжатие>` указывается число от `0` (низкое качество, маленький размер файла) до `100` (высокое качество, но большой размер файла). Значение по умолчанию — `75`;

- `imagegif()` — для изображений в формате GIF:

  ```
  imagegif(<Идентификатор>[, <Имя файла>]) : bool
  ```

- `imagepng()` — для изображений в формате PNG:

  ```
  imagepng(<Идентификатор>[, <Имя файла>[, <Сжатие>[, <Фильтры>]]]) : bool
  ```

В параметре `<Сжатие>` указывается число от `0` (без сжатия, большой размер файла) до `9` (максимальное сжатие). Значение по умолчанию — `6`. В параметре `<Фильтры>` можно указать следующие константы (или их комбинацию): `PNG_NO_FILTER` (выключение всех фильтров), `PNG_ALL_FILTERS` (включение всех фильтров), `PNG_FILTER_AVG`, `PNG_FILTER_NONE`, `PNG_FILTER_PAETH`, `PNG_FILTER_SUB` и `PNG_FILTER_UP`.

После вывода изображения следует освободить ресурсы с помощью функции `imagedestroy()`:

```
imagedestroy(<Идентификатор>)
```

В качестве примера выведем баннер banner.gif в окно Web-браузера. Для этого создадим файл banner.php (листинг 5.154).

Листинг 5.154. Файл banner.php для вывода баннера

```
<?php
// Загружаем баннер из файла
@$img = imagecreatefromgif('banner.gif');
if ( !is_resource($img) ) {
   // Создаем заглушку, если не удалось загрузить
   @$img = imagecreate(468, 60);
   $white = imagecolorallocate($img, 255, 255, 255);
}
// Удаляем заголовок X-Powered-By
header_remove('X-Powered-By');
// Отправляем заголовок
header('Content-Type: image/gif');
// Выводим изображение в Web-браузер
imagegif($img);
// Освобождаем ресурсы
imagedestroy($img);
exit();
```

Отобразить баннер в окне Web-браузера позволяет следующий HTML-код:

```
<img src="banner.php" alt="">
```

Это аналогично встраиванию обычного изображения:

```
<img src="banner.gif" alt="">
```

Но есть одно различие: если изображение содержит анимацию, то в окне Web-браузера будет отображен только первый ее кадр.

5.20.5. Сохранение изображения в файл

В функциях `imagejpeg()`, `imagegif()` и `imagepng()` необязательный параметр `<Имя файла>` задает имя файла или идентификатор потока, в который осуществляется вывод. Это означает, что изображение можно не только вывести в Web-браузер,

но и сохранить в файл. Причем, мы можем выводить или сохранять созданное или загруженное изображение в любом формате, указав соответствующую функцию при выводе.

В качестве примера загрузим изображение в формате JPEG, а затем сохраним его в формате PNG (листинг 5.155). Обратите внимание: если файл существует, то он будет перезаписан. Если не удалось записать в файл, то функции не только вернут значение `false`, но и выведут предупреждающее сообщение.

Листинг 5.155. Сохранение изображения в файл

```
<?php
@$img = imagecreatefromjpeg('photo.jpg');
if ( is_resource($img) ) {
   if (@imagepng($img, 'photo_png.png', 3)) {
      echo 'Изображение успешно сохранено';
   }
   else echo 'Не удалось сохранить изображение';
   imagedestroy($img);
}
else echo 'Не удалось загрузить изображение';
```

5.20.6. Получение информации об изображении

Получить информацию об изображении позволяют следующие функции:

- `getimagesize(<Имя файла>[, array &$imageinfo])` — возвращает информацию об изображении в виде массива:

  ```
  print_r(getimagesize('photo.jpg'));
  ```

  Результат в исходном HTML-коде:

  ```
  Array
  (
      [0] => 500
      [1] => 333
      [2] => 2
      [3] => width="500" height="333"
      [bits] => 8
      [channels] => 3
      [mime] => image/jpeg
  )
  ```

  Элемент с индексом `2` содержит тип изображения в виде значения одной из констант `IMAGETYPE_<Формат>`. Выведем значения основных констант:

  ```
  var_dump(IMAGETYPE_GIF);  // int(1)
  var_dump(IMAGETYPE_JPEG); // int(2)
  var_dump(IMAGETYPE_PNG);  // int(3)
  ```

```
var_dump(IMAGETYPE_PSD);  // int(5)
var_dump(IMAGETYPE_BMP);  // int(6)
var_dump(IMAGETYPE_ICO);  // int(17)
```

- `image_type_to_mime_type()` — возвращает MIME-тип, соответствующий константе `IMAGETYPE_<Формат>`:

```
var_dump(image_type_to_mime_type(IMAGETYPE_GIF));
// string(9) "image/gif"
var_dump(image_type_to_mime_type(IMAGETYPE_JPEG));
// string(10) "image/jpeg"
var_dump(image_type_to_mime_type(IMAGETYPE_PNG));
// string(9) "image/png"
```

- `image_type_to_extension()` — возвращает расширение файла, соответствующего константе `IMAGETYPE_<Формат>`:

```
var_dump(image_type_to_extension(IMAGETYPE_GIF));
// string(4) ".gif"
var_dump(image_type_to_extension(IMAGETYPE_JPEG));
// string(5) ".jpeg"
var_dump(image_type_to_extension(IMAGETYPE_PNG));
// string(4) ".png"
var_dump(image_type_to_extension(IMAGETYPE_PNG, false));
// string(3) "png"
```

- `imagesx(<Идентификатор>)` — возвращает ширину загруженного или созданного изображения;
- `imagesy(<Идентификатор>)` — возвращает высоту загруженного или созданного изображения:

```
@$img = imagecreatefromjpeg('photo.jpg');
// @$img = imagecreatetruecolor(500, 333);
if ( is_resource($img) ) {
   echo 'Ширина: ' . imagesx($img) . ' ';
   echo 'Высота: ' . imagesy($img);
   imagedestroy($img);
} // Выведет: Ширина: 500 Высота: 333
```

5.20.7. Библиотека *php_exif*

Получить детальную информацию о JPEG- и TIFF-изображениях позволяет библиотека php_exif.dll. Чтобы использовать эту библиотеку, нужно, чтобы в файле php.ini не было символа комментария (`;`) перед строкой:

```
extension=php_exif.dll
```

В PHP 7.2 строка выглядит следующим образом:

```
extension=exif
```

> **Внимание!**
>
> Библиотека *php_exif.dll* должна подключаться в конфигурационном файле *php.ini* после библиотеки *php_mbstring.dll*. Если это условие не соблюдается, то библиотека будет недоступна для использования.

Библиотека php_exif.dll предоставляет следующие функции:

- `exif_imagetype(<Путь к файлу>)` — позволяет определить формат файла. Функция возвращает `false`, если формат файла определить не удалось, или значение одной из констант `IMAGETYPE_<Формат>`.

  Проверить формат можно так:

  ```
  if (@exif_imagetype('photo.jpg') == IMAGETYPE_JPEG) {
     echo 'Это фото в формате JPEG';
  }
  ```

- `exif_read_data()` — позволяет вывести информацию из заголовков JPEG- и TIFF-файлов. Возвращает результат в виде ассоциативного массива. Формат функции:

  ```
  exif_read_data(<Путь к файлу>[, <Список разделов>[,
                 <Тип массива>[, <Вывод миниатюры>]]])
  ```

  В параметре `<Список разделов>` можно привести через запятую разделы, которые должны присутствовать в файле. Если указанный раздел отсутствует, то функция возвращает `false`. По умолчанию параметр имеет значение `null`.

  Параметр `<Тип массива>` определяет, будет ли каждый раздел представлен в виде отдельного массива (значение `true`) или нет (значение `false`). По умолчанию параметр имеет значение `false`. Следует учитывать, что разделы `COMPUTED`, `THUMBNAIL` и `COMMENT` всегда представлены как отдельные массивы.

  Параметр `<Вывод миниатюры>` определяет, будет ли загружена миниатюра (значение `true`) или нет (значение `false`). По умолчанию параметр имеет значение `false`.

  Вот пример использования этой функции:

  ```
  <?php
  @$arr = exif_read_data('FF5A9519.jpg');
  if ( !is_array($arr) ) die('Информации нет');
  echo '<b>Информация об изображении:</b><br>';
  echo 'Название: ' . ($arr['FileName'] ?? '') . '<br>';
  echo 'Размер: ';
  echo number_format($arr['FileSize'] ?? '0', 0, '.', ' ');
  echo '<br>';
  echo 'MIME-тип: ' . ($arr['MimeType'] ?? '') . '<br>';
  echo 'Ширина: ' . ($arr['COMPUTED']['Width'] ?? '') . '<br>';
  echo 'Высота: ' . ($arr['COMPUTED']['Height'] ?? '') . '<br>';
  echo 'Дата создания: ' . ($arr['DateTimeOriginal'] ?? '');
  echo '<br>';
  echo 'Выдержка: ' . ($arr['ExposureTime'] ?? '') . '<br>';
  ```

```
echo 'ISO: ' . ($arr['ISOSpeedRatings'] ?? '') . '<br>';
echo '<br>';
echo '<b>Информация о фотоаппарате:</b><br>';
echo 'Производитель: ' . ($arr['Make'] ?? '') . '<br>';
echo 'Модель: ' . ($arr['Model'] ?? '') . '<br>';
```

Выведет примерно:

```
Информация об изображении:
Название: FF5A9519.jpg
Размер: 2 734 965
MIME-тип: image/jpeg
Ширина: 3482
Высота: 2321
Дата создания: 2015:03:18 18:59:18
Выдержка: 1/200
ISO: 400

Информация о фотоаппарате:
Производитель: Canon
Модель: Canon EOS 5D Mark III
```

Вывести полную информацию по разделам можно следующим образом:

```
print_r(exif_read_data('FF5A9519.jpg', null, true, false));
```

- `exif_thumbnail()` — позволяет вывести миниатюру из JPEG- и TIFF-файлов. Формат функции:

```
exif_thumbnail(<Путь к файлу>[, <Ширина>[, <Высота>[,
               <Формат миниатюры>]]]) : string
```

В необязательных параметрах `<Ширина>`, `<Высота>` и `<Формат миниатюры>` можно указать переменные, в которых будут сохранены соответствующие параметры миниатюры:

```
<?php
@$img = exif_thumbnail('FF5A9519.jpg', $w, $h, $type);
if ( $img === false ) die('Миниатюры нет');
header('Content-Type: ' . image_type_to_mime_type($type));
echo $img;
exit();
```

5.20.8. Работа с цветом

Для работы с цветом предназначены следующие функции:

- `imagecolorallocate()` — добавляет новый цвет в палитру. Первый созданный цвет будет использован для заливки фона палитрового изображения. Формат функции:

```
imagecolorallocate(<Идентификатор>, <Красный>, <Зеленый>, <Синий>)
```

Компоненты цвета формата RGB задаются в виде чисел от `0` до `255`.

Вот пример создания и вывода баннера белого цвета в формате GIF:

```
<?php
@$img = imagecreate(468, 60);
$white = imagecolorallocate($img, 255, 255, 255);
header('Content-Type: image/gif');
imagegif($img);
imagedestroy($img);
exit();
```

- `imagecolorallocatealpha()` — добавляет новый цвет с альфа-каналом в палитру. Формат функции:

```
imagecolorallocatealpha(<Идентификатор>, <Красный>, <Зеленый>,
                        <Синий>, <Альфа-канал>)
```

  Компоненты цвета формата RGB задаются в виде чисел от `0` до `255`. Значение параметра `<Альфа-канал>` указывается в диапазоне от `0` (полностью непрозрачный цвет) до `127` (полностью прозрачный цвет):

```
$color = imagecolorallocatealpha($img, 0, 0, 0, 64);
```

- `imagecolorclosest()` — возвращает ближайший к указанному цвет из имеющихся в палитре:

```
imagecolorclosest(<Идентификатор>, <Красный>, <Зеленый>, <Синий>)
```

  Пример:

```
$color = imagecolorclosest($img, 255, 255, 255);
```

- `imagecolordeallocate()` — уничтожает объект цвета, созданный функциями `imagecolorallocate()` и `imagecolorallocatealpha()`. Формат функции:

```
imagecolordeallocate(<Идентификатор>, <Цвет>)
```

- `imagecolortransparent()` — делает указанный цвет палитры прозрачным:

```
imagecolortransparent(<Идентификатор>, <Цвет>)
```

  Вот пример создания и вывода GIF-файла с прозрачным фоном и синим квадратом:

```
<?php
@$img = imagecreate(468, 60);
// Задаем цвет фона
$black = imagecolorallocate($img, 0, 0, 0);
// Делаем фон прозрачным
imagecolortransparent($img, $black);
$blue = imagecolorallocate($img, 0, 0, 255);
imagefilledrectangle($img, 10, 10, 40, 40, $blue);
header('Content-Type: image/gif');
imagegif($img);
imagedestroy($img);
exit();
```

- `imagefill()` — позволяет закрасить область одного цвета другим цветом. Достаточно указать координаты точки (начало координат расположено в левом верхнем углу изображения) и новый цвет:

```
imagefill(<Идентификатор>, <X>, <Y>, <Цвет>)
```

- `imagefilltoborder()` — позволяет закрасить область, ограниченную точками какого-то цвета, другим цветом. Достаточно указать координаты точки, а также цвета границы и заливки:

```
imagefilltoborder(<Идентификатор>, <X>, <Y>, <Цвет границы>, <Цвет заливки>)
```

- `imagecolorat()` — возвращает индекс цвета указанной точки палитрового изображения:

```
imagecolorat(<Идентификатор>, <X>, <Y>)
```

Если изображение является полноцветным, то функция возвращает целочисленное значение RGB. Получить значения отдельных компонентов цвета в этом случае можно так:

```
@$img = imagecreatefromjpeg('photo.jpg');
$color = imagecolorat($img, 0, 0);
$r = ($color >> 16) & 0xFF;
$g = ($color >> 8) & 0xFF;
$b = $color & 0xFF;
var_dump($r, $g, $b);
```

- `imagecolorsforindex()` — возвращает ассоциативный массив значений компонентов цвета для указанного идентификатора цвета или целочисленного значения RGB:

```
imagecolorsforindex(<Идентификатор изображения>, <Цвет>)
```

В качестве примера выведем числовые значения компонентов цвета указанной точки изображения:

```
@$img = imagecreatefromjpeg('photo.jpg');
$color = imagecolorat($img, 0, 0);
print_r(imagecolorsforindex($img, $color));
imagedestroy($img);
```

В результате получим:

```
Array
(
    [red] => 178
    [green] => 174
    [blue] => 162
    [alpha] => 0
)
```

- `imagecolorstotal()` — возвращает число цветов в палитре изображения:

```
imagecolorstotal(<Идентификатор>)
```

Пример:

```
@$img = imagecreatefromgif('banner.gif');
print_r(imagecolorstotal($img));
imagedestroy($img);
```

Создадим полноцветное изображение с прозрачным фоном, нарисуем на нем прямоугольник красного цвета и выведем его в формате PNG (листинг 156).

Листинг 5.156. Создание изображения с прозрачным фоном

```
<?php
@$img = imagecreatetruecolor(468, 60);
// Включаем сохранение информации о прозрачности
imagesavealpha($img, true);
// Делаем фон прозрачным
$transparent = imagecolorallocatealpha($img, 0, 0, 0, 127);
imagefill($img, 0, 0, $transparent);
// Рисуем прямоугольник красного цвета
$color = imagecolorallocatealpha($img, 255, 0, 0, 0);
imagefilledrectangle($img, 10, 10, 150, 40, $color);
// Выводим в Web-браузер
header('Content-Type: image/png');
imagepng($img);
imagedestroy($img);
exit();
```

Обратите внимание на следующую инструкцию:

```
imagesavealpha($img, true);
```

По умолчанию при сохранении или выводе изображения информация об альфа-канале удаляется, в результате чего фон будет черного цвета, а не прозрачным. Чтобы этого не происходило при работе с изображениями в формате PNG, содержащими альфа-канал, нужно обязательно вызвать функцию `imagesavealpha()` и передать ей значение `true`.

5.20.9. Смешивание цветов

Если мы рисуем фигуру сплошным цветом без альфа-канала, то новые пикселы просто заменят старые. Если же в процессе наложения участвуют цвета с альфа-каналом, то возможны варианты. По умолчанию при использовании полноцветных изображений цвета будут смешиваться. Изменить поведение по умолчанию позволяет функция `imagealphablending()`:

```
imagealphablending(<Идентификатор>, <true | false>)
```

Рассмотрим работу функции на примере (листинг 5.157).

Листинг 5.157. Смешивание цветов

```
<?php
@$img = imagecreatetruecolor(468, 60);
imagealphablending($img, true);
imagesavealpha($img, true);
$bg = imagecolorallocatealpha($img, 255, 255, 255, 20);
imagefill($img, 0, 0, $bg);
$color = imagecolorallocatealpha($img, 255, 0, 0, 50);
imagefilledrectangle($img, 0, 0, 150, 40, $color);
print_r(imagecolorsforindex($img, imagecolorat($img, 0, 0)));
imagedestroy($img);
```

В этом примере мы создали полупрозрачный фон и нарисовали прямоугольник полупрозрачным красным цветом. Если в функции `imagealphablending()` второй параметр имеет значение `true` или функция вообще не вызвана (при использовании полноцветных изображений), то цвета будут взаимодействовать друг с другом, и мы получим следующий результат после смешивания:

```
// imagealphablending($img, true);
// Array ( [red] => 255 [green] => 90 [blue] => 90 [alpha] => 7 )
```

Если в функции `imagealphablending()` второй параметр имеет значение `false`, то никакого смешивания не произойдет, и новые пикселы просто заменят старые:

```
// imagealphablending($img, false);
// Array ( [red] => 255 [green] => 0 [blue] => 0 [alpha] => 50 )
```

5.20.10. Рисование линий и фигур

Библиотека GD позволяет рисовать следующие фигуры:

- точка:

  ```
  imagesetpixel(<Идентификатор>, <X>, <Y>, <Цвет>)
  ```

  Здесь `<X>` и `<Y>` — координаты точки относительно левого верхнего угла;
- сплошная линия:

  ```
  imageline(<Идентификатор>, <X1>, <Y1>, <X2>, <Y2>, <Цвет>)
  ```

  Линия задается двумя точками с координатами (`<X1>`,`<Y1>`) и (`<X2>`,`<Y2>`);
- прямоугольник без заливки:

  ```
  imagerectangle(<Идентификатор>, <X1>, <Y1>, <X2>, <Y2>, <Цвет обводки>)
  ```

  - `<X1>` и `<Y1>` — координаты левого верхнего угла;
  - `<X2>` и `<Y2>` — координаты правого нижнего угла;
  - `<Цвет обводки>` — цвет границы;

- прямоугольник с заливкой:

```
imagefilledrectangle(<Идентификатор>, <X1>, <Y1>, <X2>, <Y2>,
                     <Цвет заливки>)
```

  - <X1> и <Y1> — координаты левого верхнего угла;
  - <X2> и <Y2> — координаты правого нижнего угла;
  - <Цвет заливки> — цвет прямоугольника;
- эллипс без заливки:

```
imageellipse(<Идентификатор>, <X>, <Y>, <Ширина>, <Высота>, <Цвет обводки>)
```

  - <X> и <Y> — координаты центра;
  - <Ширина> и <Высота> — размеры;
  - <Цвет обводки> — цвет границы;
- эллипс с заливкой:

```
imagefilledellipse(<Идентификатор>, <X>, <Y>, <Ширина>,
                   <Высота>, <Цвет заливки>)
```

  - <X> и <Y> — координаты центра;
  - <Ширина> и <Высота> — размеры;
  - <Цвет заливки> — цвет эллипса;
- многоугольник без заливки:

```
imagepolygon(<Идентификатор>, <Массив координат>,
             <Число вершин>, <Цвет обводки>)
```

  - <Массив координат> — массив координат вершин;
  - <Число вершин> — число вершин многоугольника;
  - <Цвет обводки> — цвет границы.

  Пример рисования треугольника:

```
$arr = [10, 80, 40, 10, 80, 80];
imagepolygon($img, $arr, 3, $color);
```

- многоугольник с заливкой:

```
imagefilledpolygon(<Идентификатор>, <Массив координат>,
                   <Число вершин>, <Цвет заливки>)
```

  - <Массив координат> — массив координат вершин;
  - <Число вершин> — число вершин многоугольника;
  - <Цвет заливки> — цвет многоугольника;
- дуга:

```
imagearc(<Идентификатор>, <X>, <Y>, <Ширина>, <Высота>,
         <Начало>, <Конец>, <Цвет обводки>)
```

- <X> и <Y> — координаты центра;
- <Ширина> и <Высота> — размеры;
- <Начало> — начальный угол в градусах;
- <Конец> — конечный угол в градусах. Угол 0° соответствует положению 3 часа, углы отсчитываются по часовой стрелке;
- <Цвет обводки> — цвет границы.

Пример рисования дуги:

```
<?php
@$img = imagecreate(200, 200);
if (!is_resource($img)) die('Ошибка');
$white = imagecolorallocate($img, 255, 255, 255);
$red = imagecolorallocate($img, 255, 0, 0);
imagearc($img, 100, 100, 150, 150, 0, 270, $red);
header('Content-Type: image/gif');
imagegif($img);
imagedestroy($img);
```

- дуга или сектор:

```
imagefilledarc(<Идентификатор>, <X>, <Y>, <Ширина>, <Высота>,
               <Начало>, <Конец>, <Цвет>, <Стиль>)
```

- <X> и <Y> — координаты центра;
- <Ширина> и <Высота> — размеры;
- <Начало> — начальный угол в градусах;
- <Конец> — конечный угол в градусах. Угол 0° соответствует положению 3 часа, углы отсчитываются по часовой стрелке;
- <Цвет> — цвет заливки или обводки;
- <Стиль> — стиль фигуры:
  - IMG_ARC_PIE — соединяет концы дуги частью окружности. Если дополнительно указан флаг IMG_ARC_NOFILL, то рисуется просто дуга. Если флаг IMG_ARC_NOFILL не указан, то получим сектор с заливкой;
  - IMG_ARC_CHORD — соединяет концы дуги прямой линией;
  - IMG_ARC_NOFILL — предписывает использовать характеристики обводки без заливки. Заливка используется по умолчанию;
  - IMG_ARC_EDGED — если флаг указан вместе с IMG_ARC_NOFILL, то получим сектор с обводкой без заливки.

Нарисуем сектор окружности с заливкой и без заливки, а также дугу:

```
<?php
@$img = imagecreate(200, 200);
if (!is_resource($img)) die('Ошибка');
```

```
$white = imagecolorallocate($img, 255, 255, 255);
$red = imagecolorallocate($img, 255, 0, 0);
imagefilledarc($img, 100, 100, 150, 150, 0, 90, $red,
               IMG_ARC_PIE);
imagefilledarc($img, 100, 100, 150, 150, 180, 270, $red,
               IMG_ARC_EDGED | IMG_ARC_NOFILL);
imagefilledarc($img, 100, 100, 150, 150, 290, 340, $red,
               IMG_ARC_PIE | IMG_ARC_NOFILL);
header('Content-Type: image/gif');
imagegif($img);
imagedestroy($img);
```

5.20.11. Изменение характеристик линии

Изменить характеристики линии позволяют следующие функции:

- `imagesetthickness()` — устанавливает толщину линий при рисовании:

  ```
  imagesetthickness(<Идентификатор>, <Толщина в пикселах>)
  ```

  По умолчанию толщина линий составляет 1 пиксел;

- `imagesetstyle()` — задает стиль линии при рисовании, если в качестве параметра `<Цвет>` в функциях для рисования указана константа `IMG_COLOR_STYLED`:

  ```
  imagesetstyle(<Идентификатор>, <Массив цветов пикселов>)
  ```

  Пример:

  ```
  <?php
  @$img = imagecreate(200, 200);
  if (!is_resource($img)) die('Ошибка');
  $white = imagecolorallocate($img, 255, 255, 255);
  $black = imagecolorallocate($img, 0, 0, 0);
  $style = array($black, $black, $black, $white, $white,
                 $white, $white);
  imagesetstyle($img, $style);
  imageline($img, 20, 100, 180, 100, IMG_COLOR_STYLED);
  header('Content-Type: image/gif');
  imagegif($img);
  imagedestroy($img);
  ```

- `imagesetbrush()` — позволяет задать изображение, которое будет использовано в качестве кисти, если в параметре `<Цвет>` в функциях для рисования указана константа `IMG_COLOR_BRUSHED`:

  ```
  imagesetbrush(<Идентификатор>, <Идентификатор кисти>)
  ```

  Вот пример использования смайлика в качестве кисти:

  ```
  @$brush = imagecreatefrompng('smile.png');
  if (!is_resource($brush)) die('Ошибка');
  imagesetbrush($img, $brush);
  imageline($img, 20, 100, 180, 100, IMG_COLOR_BRUSHED);
  ```

- imageantialias() — управляет режимом сглаживания линий. Значение true включает сглаживание, а значение false — отключает. Формат функции:

```
imageantialias(<Идентификатор>, <true | false>)
```

  Обратите внимание: функция imageantialias() работает только с полноцветными изображениями и не поддерживает альфа-канал, а также толщину и стиль линии:

```
imageantialias($img, true);
imageline($img, 40, 40, 180, 150, $red);
```

5.20.12. Вывод текста в изображение

Для вывода текста используются следующие функции:

- imagechar() — рисует символ на изображении по горизонтали:

```
imagechar(<Идентификатор>, <Шрифт>, <X>, <Y>, <Символ>, <Цвет>)
```

- imagecharup() — рисует символ на изображении по вертикали:

```
imagecharup(<Идентификатор>, <Шрифт>, <X>, <Y>, <Символ>, <Цвет>)
```

- imagestring() — отображает строку на изображении по горизонтали:

```
imagestring(<Идентификатор>, <Шрифт>, <X>, <Y>, <Строка>, <Цвет>)
```

- imagestringup() — отображает строку на изображении по вертикали:

```
imagestringup(<Идентификатор>, <Шрифт>, <X>, <Y>, <Строка>, <Цвет>)
```

В этих функциях параметр <Шрифт> задает размер встроенного шрифта, который выражается числом от 1 до 5.

Все четыре функции с буквами русского языка не работают. Для русского языка следует применять TrueType-шрифты (например, arial.ttf). В Windows шрифты расположены в каталоге C:\Windows\Fonts. Для работы с TrueType-шрифтами предназначены следующие функции:

- imagettftext() — рисует строку на изображении TrueType-шрифтом. Функция возвращает массив координат четырех вершин прямоугольника, в который будет вписан текст. Вершины указываются в следующем порядке: нижняя левая, нижняя правая, верхняя правая, верхняя левая. Формат функции:

```
imagettftext(<Идентификатор>, <Размер>, <Угол>, <X>, <Y>,
             <Цвет>, <Шрифт>, <Строка>)
```

  Параметры имеют следующий смысл:

  - <X> и <Y> — координаты левой крайней точки базовой линии;
  - <Размер> — размер шрифта;
  - <Угол> — угол поворота текста. Угол 0° соответствует обычному горизонтальному расположению текста, а поворот осуществляется против часовой стрелки;
  - <Шрифт> — имя файла со шрифтом.

Выведем текст и обведем его рамкой:

```
<?php
@$img = imagecreate(400, 100);
if (!is_resource($img)) die('Ошибка');
$white = imagecolorallocate($img, 255, 255, 255);
$red = imagecolorallocate($img, 255, 0, 0);
$black = imagecolorallocate($img, 0, 0, 0);
$font = 'C:\\Windows\\Fonts\\arial.ttf';
$a = imagettftext($img, 20, 0, 11, 21, $red, $font,
                  'Текст на русском языке');
imagerectangle($img, $a[6], $a[7], $a[2], $a[3], $black);
header('Content-Type: image/gif');
imagegif($img);
imagedestroy($img);
```

❒ `imagettfbbox()` — возвращает координаты прямоугольника, в который вписана строка с помощью TrueType-шрифта. Формат функции:

```
imagettfbbox(<Размер>, <Угол>, <Шрифт>, <Строка>)
```

Например, такой код:

```
$font = 'C:\\Windows\\Fonts\\arial.ttf';
$arr = imagettfbbox(20, 0, $font, 'Текст на русском языке');
print_r($arr);
```

выведет:

```
Array ( [0] => 0 [1] => 5 [2] => 297 [3] => 5 [4] => 297
        [5] => -20 [6] => 0 [7] => -20 )
```

Это означает, что заданный текст, выведенный под заданным углом таким шрифтом и размером, поместится в прямоугольник со следующими координатами:

- `0`, `5` — левый нижний угол;
- `297`, `5` — правый нижний угол;
- `297`, `-20` — правый верхний угол;
- `0`, `-20` — левый верхний угол.

Внимание!

Две координаты по `Y` имеют отрицательные значения. Это происходит потому, что началом координат считается левая точка базовой линии. *Базовая линия* — это линия, соприкасающаяся с большинством букв снизу. Части некоторых букв могут быть расположены ниже базовой линии (например, буква «у»). Все, что расположено ниже базовой линии, имеет положительные координаты `Y`, а все, что выше, — отрицательные. Кроме того, отрицательные значения может иметь и координата `X`.

При выводе текста на готовое изображение возможны проблемы с цветом текста. Особенно это характерно для изображений в формате GIF, т. к. количество цветов

в палитре ограничено числом 256. Если попытаться добавить новый цвет при максимальном количестве цветов в палитре, функция `imagecolorallocate()` вместо идентификатора цвета вернет значение `false`, а цвет текста будет соответствовать цвету фона. Один из способов решения этой проблемы заключается в использовании функции `imagecolorclosest()`, которая возвращает ближайший цвет, имеющийся в палитре (листинг 5.158).



Рис. 5.2. Вывод текста на готовое изображение

Листинг 5.158. Вывод текста на готовое изображение

```
<?php
@$img = imagecreatefromgif('foto.gif');
if ( !is_resource($img) ) die('Ошибка');
header('Content-Type: image/gif');
$white = imagecolorallocate($img, 255, 255, 255);
if ($white !== false) {
   $str = 'Белый цвет';
}
```

```
else {
   $white2 = imagecolorclosest($img, 255, 255, 255);
   $str = 'Ближайший цвет';
}
$font = 'C:/Windows/Fonts/arial.ttf';
imagettftext($img, 28, 0, 100, 400, $white2, $font, $str);
$str = 'Точный цвет';
imagettftext($img, 28, 0, 140, 440, $white, $font, $str);
imagegif($img);
imagedestroy($img);
exit();
```

Здесь мы выводим на изображение две строки. Первая строка выводится цветом, ближайшим к указанному цвету в палитре. Вторая строка демонстрирует цвет текста, который получится, если бы мы не использовали функцию `imagecolorclosest()`. Результат выполнения листинга 5.158 изображен на рис. 5.2.

5.20.13. Создаем счетчик посещений

В качестве примера создадим счетчик посещения с использованием cookies и выведем результат в графическом виде (листинг 5.159).

Листинг 5.159. Счетчик посещений

```
<?php
$count = $_COOKIE['page_views'] ?? 0;
$count++;
setcookie('page_views', $count, time() + 60 * 60 * 24 * 365);
header('Content-Type: image/gif');
$img = imagecreate(88, 31);
$white = imagecolorallocate($img, 255, 255, 255);
$gray = imagecolorallocate($img, 128, 128, 128);
$black = imagecolorallocate($img, 0, 0, 0);
imagerectangle($img, 0, 0, 87, 30, $black);
$font = 'C:/Windows/Fonts/arial.ttf';
$str = 'Мой счетчик';
imagettftext($img, 8, 0, 11, 13, $gray, $font, $str);
$arr = imagettfbbox(12, 0, $font, $count);
$indent = intval((88 - $arr[2]) / 2);
imagettftext($img, 12, 0, $indent, 27, $gray, $font, $count);
imagegif($img);
imagedestroy($img);
exit();
```

Попробуйте обновить страницу — цифры на счетчике будут увеличиваться.

5.20.14. Изменение размеров и копирование изображений

Для изменения размеров и копирования изображений применяется функция `imagecopyresampled()`. Формат функции:

```
imagecopyresampled(<Идентификатор1>, <Идентификатор2>, <X1>, <Y1>,
                   <X2>, <Y2>, <Ширина1>, <Высота1>, <Ширина2>,
                   <Высота2>)
```

Параметр `<Идентификатор1>` задает изображение, в которое требуется скопировать изображение, заданное параметром `<Идентификатор2>`. Изображение, в которое осуществляется копирование, должно быть полноцветным. Для создания подложки при изменении размеров изображения часто применяется функция `imagecreatetruecolor()`.

Параметры `<X2>`, `<Y2>`, `<Ширина2>` и `<Высота2>` задают прямоугольную область в изображении, заданном в параметре `<Идентификатор2>`, которую мы будем копировать. А параметры `<X1>`, `<Y1>`, `<Ширина1>` и `<Высота1>` определяют прямоугольную область, в которую будет вставлен копируемый фрагмент.

Вместо функции `imagecopyresampled()` можно использовать функцию `imagecopyresized()`, имеющую точно такой же формат, но результат будет хуже.

В качестве примера уменьшим изображение в два раза и выведем полученное изображение в Web-браузер (листинг 5.160).

Листинг 5.160. Изменение размера изображения

```
<?php
@$img = imagecreatefromjpeg('photo.jpg');
if ( !is_resource($img) ) die('Ошибка');
// Получаем размеры изображения
$width = imagesx($img);
$height = imagesy($img);
// Получаем ширину и высоту нового изображения
$w = intval($width / 2);
$h = intval($height / 2);
// Создаем подложку для нового изображения
@$img2 = imagecreatetruecolor($w, $h);
if (!is_resource($img2)) die('Не удалось создать');
// Копируем и изменяем размер
imagecopyresampled($img2, $img, 0, 0, 0, 0,
                   $w, $h, $width, $height);
// Выводим изображение
header('Content-Type: image/jpeg');
imagejpeg($img2, null, 100);
imagedestroy($img);
imagedestroy($img2);
exit();
```

В листинге 5.158 мы рассмотрели проблему вывода текста на готовое изображение и не получили точный цвет текста. Теперь реализуем вывод текста указанным цветом. Для этого загружаем исходное изображение, создаем подложку такого же размера и копируем исходное изображение на созданную подложку. После чего выводим надпись нужным цветом (листинг 5.161).

Листинг 5.161. Вывод текста на готовое изображение указанным цветом

```
<?php
@$img = imagecreatefromgif('foto.gif');
if ( !is_resource($img) ) die('Ошибка');
// Получаем размеры изображения
$width = imagesx($img);
$height = imagesy($img);
// Создаем подложку для нового изображения
@$img2 = imagecreatetruecolor($width, $height);
if (!is_resource($img2)) die('Не удалось создать');
// Копируем исходное изображение на подложку
imagecopyresampled($img2, $img, 0, 0, 0, 0, $width, $height,
                   $width, $height);
imagedestroy($img);
$white = imagecolorallocate($img2, 255, 255, 255);
if ($white === false) die('Ошибка');
$str = 'Фонтан Самсон';
$font = 'C:/Windows/Fonts/arial.ttf';
// Выводим надпись на изображение
imagettftext($img2, 28, 0, 100, 440, $white, $font, $str);
imagetruecolortopalette($img2, false, 256);
// Выводим изображение
header('Content-Type: image/gif');
imagegif($img2);
imagedestroy($img2);
exit();
```

Результат выполнения этого скрипта показан на рис. 5.3.

Функция `imagecopymerge()` выполняет копирование фрагмента изображения с наложением:

```
imagecopymerge(<Идентификатор1>, <Идентификатор2>, <X1>, <Y1>,
               <X2>, <Y2>, <Ширина>, <Высота>, <Наложение>)
```

Последний параметр указывает степень наложения копируемого фрагмента на целевое изображение в виде числа от `0` (без наложения) до `100` (полное наложение копируемого фрагмента на целевое изображение).

Если нужно просто скопировать фрагмент, то можно воспользоваться функцией `imagecopy()`. Однако следует учитывать, что она не поддерживает прозрачность. Формат функции:

```
imagecopy(<Идентификатор1>, <Идентификатор2>, <X1>, <Y1>,
          <X2>, <Y2>, <Ширина>, <Высота>)
```

Изменить масштаб изображения с указанием метода интерполяции позволяет функция `imagescale()`. Формат функции:

```
imagescale(<Идентификатор>, int $new_width[, int $new_height=-1[,
           int $mode=IMG_BILINEAR_FIXED]]) : resource
```

Во втором параметре указывается новая ширина, а в третьем — новая высота. Если вместо высоты указано отрицательное значение или параметр не указан, то значение параметра будет рассчитано автоматически с соблюдением пропорций. Параметр `$mode` задает метод интерполяции, который будет использоваться при изменении масштаба. Можно указать константы `IMG_BILINEAR_FIXED`, `IMG_BICUBIC`, `IMG_BICUBIC_FIXED`, `IMG_NEAREST_NEIGHBOUR` и др. Функция возвращает новое изображение:

```
@$img2 = imagescale($img, 300, -1, IMG_BILINEAR_FIXED);
```



Рис. 5.3. Вывод текста на готовое изображение определенным цветом

5.20.15. Обрезка изображения

Функция `imagecrop()` позволяет выполнить обрезку изображения. Формат функции:

```
imagecrop(<Идентификатор>, <Массив>) : resource
```

Во втором параметре указывается ассоциативный массив с координатами и размерами прямоугольной области. Функция возвращает новое изображение. Пример:

```
@$img = imagecreatefromjpeg('photo.jpg');
if ( !is_resource($img) ) die('Ошибка');
@$img2 = imagecrop($img, ['x' => 50, 'y' => 50,
                   'width' => 100, 'height' => 100]);
if ( !is_resource($img2) ) die('Ошибка');
header('Content-Type: image/jpeg');
imagejpeg($img2, null, 100);
```

5.20.16. Вращение изображения

Повернуть изображение на некоторый угол относительно центра позволяет функция `imagerotate()`. Формат функции:

```
imagerotate(<Идентификатор>, <Угол>, <Цвет фона>) : resource
```

Результат работы функции зависит от используемого метода интерполяции, который задается с помощью функции `imagesetinterpolation()`:

```
imagesetinterpolation(<Идентификатор>[, <Метод>=IMG_BILINEAR_FIXED])
```

По умолчанию используется билинейная интерполяция. Полный список констант, задающих метод интерполяции, смотрите в документации.

Пример поворота изображения на 45 градусов приведен в листинге 5.162.

Листинг 5.162. Вращение изображения

```
<?php
@$img = imagecreatefromjpeg('photo.jpg');
if ( !is_resource($img) ) die('Ошибка');
$transparent = imagecolorallocatealpha($img, 0, 0, 0, 127);
imagesetinterpolation($img, IMG_BILINEAR_FIXED);
@$img2 = imagerotate($img, 45, $transparent);
if ( !is_resource($img2) ) die('Ошибка');
imagesavealpha($img2, true);
header('Content-Type: image/png');
imagepng($img2);
imagedestroy($img);
imagedestroy($img2);
exit();
```

5.20.17. Аффинные преобразования

Функция `imageaffine()` позволяет выполнить различные трансформации изображения (смещение, масштабирование, вращение и сдвиг). Формат функции:

```
imageaffine(<Идентификатор>, <Матрица>[, array $clip]) : resource
```

В параметре `<Матрица>` указывается массив из шести элементов:

```
$matrix = [ mxx, mxy, myx, myy, tx, ty ];
```

Матрица трансформации имеет следующий вид:

```
mxx  mxy  tx
myx  myy  ty
```

Вычисление координат с учетом трансформации производится так:

```
x = mxx * x + mxy * y + tx
y = myx * x + myy * y + ty
```

Вот пример матрицы трансформации для масштабирования в два раза:

```
$matrix = [ 2, 0, 0, 2, 0, 0 ];
```

Создать матрицу трансформации позволяет функция `imageaffinematrixget()`. Формат функции:

```
imageaffinematrixget(<Опция>, <Значение>) : array
```

Если в первом параметре задана константа `IMG_AFFINE_TRANSLATE` (смещение) или `IMG_AFFINE_SCALE` (масштаб), то во втором параметре должен быть указан ассоциативный массив с ключами `x` и `y`:

```
$matrix = imageaffinematrixget(IMG_AFFINE_TRANSLATE,
                               ['x' => 10, 'y' => 20]);
print_r($matrix);
// Array ( [0] => 1 [1] => 0 [2] => 0 [3] => 1 [4] => 10 [5] => 20 )
$matrix = imageaffinematrixget(IMG_AFFINE_SCALE,
                               ['x' => 2, 'y' => 1.5]);
print_r($matrix);
// Array ( [0] => 2 [1] => 0 [2] => 0 [3] => 1.5 [4] => 0 [5] => 0 )
```

Если в первом параметре задана константа `IMG_AFFINE_ROTATE` (вращение), `IMG_AFFINE_SHEAR_HORIZONTAL` (сдвиг по горизонтали) или `IMG_AFFINE_SHEAR_VERTICAL` (сдвиг по вертикали), то во втором параметре должно быть указано значение в градусах:

```
$matrix = imageaffinematrixget(IMG_AFFINE_ROTATE, 45);
print_r($matrix);
// Array ( [0] => 0.70710678118655 [1] => 0.70710678118655
// [2] => -0.70710678118655 [3] => 0.70710678118655
// [4] => 0 [5] => 0 )
```

С помощью функции `imageaffinematrixconcat()` можно объединить две матрицы трансформации в одну. Формат функции:

```
imageaffinematrixconcat(array $m1, array $m2) : array
```

Уменьшим размер изображения в два раза и повернем его на 45 градусов против часовой стрелки (листинг 5.163).

Листинг 5.163. Аффинные преобразования

```
<?php
@$img = imagecreatefromjpeg('photo.jpg');
if ( !is_resource($img) ) die('Ошибка');
imagesetinterpolation($img, IMG_BILINEAR_FIXED);
$m1 = imageaffinematrixget(IMG_AFFINE_ROTATE, -45);
$m2 = imageaffinematrixget(IMG_AFFINE_SCALE,
                           ['x' => 0.5, 'y' => 0.5]);
$matrix = imageaffinematrixconcat($m1, $m2);
@$img2 = imageaffine($img, $matrix);
if ( !is_resource($img2) ) die('Ошибка');
imagesavealpha($img2, true);
header('Content-Type: image/png');
imagepng($img2);
imagedestroy($img);
imagedestroy($img2);
exit();
```

5.20.18. Применение фильтров

Функция `imagefilter()` позволяет применить фильтр к изображению. Формат функции:

```
imagefilter(<Идентификатор>, <Фильтр>[, int $arg1[, int $arg2[,
            int $arg3[, int $arg4]]]]) : bool
```

Размытие изображения

Размыть или сгладить изображение позволяют следующие фильтры:

- `IMG_FILTER_GAUSSIAN_BLUR` — размытие методом Гаусса:

  ```
  imagefilter($img, IMG_FILTER_GAUSSIAN_BLUR);
  ```

- `IMG_FILTER_SELECTIVE_BLUR` — размывает изображение:

  ```
  imagefilter($img, IMG_FILTER_SELECTIVE_BLUR);
  ```

- `IMG_FILTER_SMOOTH` — сглаживает изображение. Степень сглаживания задается в параметре `$arg1`:

  ```
  imagefilter($img, IMG_FILTER_SMOOTH, 3);
  ```

Изменение яркости и контраста

Изменить яркость или контраст позволяют следующие фильтры:

- IMG_FILTER_BRIGHTNESS — изменяет яркость изображения. Уровень яркости задается в параметре $arg1 (от –255 до 255):

  ```
  imagefilter($img, IMG_FILTER_BRIGHTNESS, 10);
  ```

- IMG_FILTER_CONTRAST — изменяет контраст изображения. Уровень контраста задается в параметре $arg1 (от –100 до 100):

  ```
  imagefilter($img, IMG_FILTER_CONTRAST, -10);
  ```

Изменение цвета

Преобразовать цветное изображение в изображение в градациях серого цвета можно с помощью фильтра IMG_FILTER_GRAYSCALE:

```
imagefilter($img, IMG_FILTER_GRAYSCALE);
```

Для преобразования изображения к оттенкам какого-либо цвета предназначен фильтр IMG_FILTER_COLORIZE. Значения компонентов цвета RGB задаются в параметрах $arg1, $arg2 и $arg3 (от 0 до 255), а прозрачность в параметре $arg4 (от 0 до 127):

```
imagefilter($img, IMG_FILTER_COLORIZE, 255, 0, 0, 50);
```

Инвертировать значения компонентов цвета позволяет фильтр IMG_FILTER_NEGATE:

```
imagefilter($img, IMG_FILTER_NEGATE);
```

Выделение границ

Фильтр IMG_FILTER_EDGEDETECT позволяет выделить границы объектов:

```
imagefilter($img, IMG_FILTER_EDGEDETECT);
```

Сделать изображение рельефным можно с помощью фильтра IMG_FILTER_EMBOSS:

```
imagefilter($img, IMG_FILTER_EMBOSS);
```

Фильтр IMG_FILTER_MEAN_REMOVAL реализует эффект «эскиза»:

```
imagefilter($img, IMG_FILTER_MEAN_REMOVAL);
```

Разделение изображения на блоки

Разделить изображение на отдельные одноцветные блоки позволяет фильтр IMG_FILTER_PIXELATE. Размер блока указывается в параметре $arg1, а в параметре $arg2 можно задать значения true или false (по умолчанию):

```
imagefilter($img, IMG_FILTER_PIXELATE, 5, true);
```

Применение произвольного фильтра

Применить к изображению фильтр с произвольным ядром свертки размером 3×3 позволяет функция `imageconvolution()`. Формат функции:

```
imageconvolution(<Идентификатор>, array $matrix, float $div,
                 float $offset) : bool
```

В первом параметре указывается матрица ядра фильтра в виде многомерного массива, во втором — коэффициент нормализации, а в третьем — смещение цвета.

Приведем несколько примеров:

- размытие методом однородного сглаживания:

  ```
  $kernel = [
     [1, 1, 1],
     [1, 1, 1],
     [1, 1, 1]
  ];
  imageconvolution($img, $kernel, 9, 0);
  ```

- увеличение резкости:

  ```
  $kernel = [
     [ 0.1111, -0.8889,  0.1111],
     [-0.8889,  4.1111, -0.8889],
     [ 0.1111, -0.8889,  0.1111]
  ];
  imageconvolution($img, $kernel, 1, 0);
  ```

- выделение границ:

  ```
  $kernel = [
     [ 0,  1,  0],
     [ 1, -4,  1],
     [ 0,  1,  0]
  ];
  imageconvolution($img, $kernel, 1, 0);
  ```

5.20.19. Создание зеркального отражения

Создать зеркальное отражение по горизонтали, вертикали или по горизонтали и по вертикали позволяет функция `imageflip()`:

```
imageflip(<Идентификатор>, <Режим>) : bool
```

Параметр `<Режим>` задает направление отражения:

- `IMG_FLIP_HORIZONTAL` — отражение по горизонтали;
- `IMG_FLIP_VERTICAL` — отражение по вертикали;
- `IMG_FLIP_BOTH` — отражение и по горизонтали, и по вертикали.

Пример:

```
imageflip($img, IMG_FLIP_HORIZONTAL);
```

5.20.20. Создание скриншота экрана

Создать скриншот экрана позволяет функция `imagegrabscreen()`. Функция возвращает изображение или значение `false` — в случае неудачи:

```
<?php
@$img = imagegrabscreen();
if ( !is_resource($img) ) die('Ошибка');
header('Content-Type: image/jpeg');
imagejpeg($img, null, 100);
imagedestroy($img);
exit();
```

5.21. Другие полезные функции

В этом разделе мы рассмотрим дополнительные функции, которые могут пригодиться при написании скриптов на PHP.

5.21.1. Выделение фрагментов исходного кода

С помощью функции `highlight_file()` можно подсветить синтаксис PHP-кода. В качестве параметра нужно передать имя файла с PHP-кодом, и в результате получим содержимое файла с выделением синтаксиса в окне Web-браузера или в виде строки (если второй параметр имеет значение `true`). Формат функции:

```
highlight_file(<Имя файла>[, <true | false>=false])
```

Пример:

```
highlight_file('banner.php');
$str = highlight_file('banner.php', true);
echo $str;
```

Функция `highlight_string()` позволяет подсветить PHP-код, находящийся в строке. В результате получим содержимое строки с выделением синтаксиса в окне Web-браузера или в виде строки (если второй параметр имеет значение `true`). Формат функции:

```
highlight_string(<Имя файла>[, <true | false>=false])
```

Пример:

```
$str = '<?php $s = "строка"; ?>';
highlight_string($str);
$str = highlight_string($str, true);
echo $str;
```

Управлять цветами можно с помощью следующих директив в файле php.ini:

```
highlight.string  = #DD0000
highlight.comment = #FF9900
highlight.keyword = #007700
highlight.default = #0000BB
highlight.html    = #000000
```

5.21.2. Получение информации об интерпретаторе

Для этого предназначены следующие функции:

- `phpinfo([<Раздел>])` — возвращает детальную информацию об интерпретаторе:

```
<?php
phpinfo();
```

- `phpversion([<Расширение>])` — служит для определения версии интерпретатора:

```
echo phpversion(); // Выведет: 7.2.0
```

  Можно также воспользоваться следующими константами:

```
var_dump(PHP_VERSION);         // string(5) "7.2.0"
var_dump(PHP_VERSION_ID);      // int(70200)
var_dump(PHP_MAJOR_VERSION);   // int(7)
var_dump(PHP_MINOR_VERSION);   // int(2)
var_dump(PHP_RELEASE_VERSION); // int(0)
var_dump(PHP_EXTRA_VERSION);   // string(0) ""
```

- `version_compare()` — сравнивает две версии. Формат функции:

```
version_compare(string $version1, string $version2[,
                string $operator]) : mixed
```

  Вот пример проверки использования PHP 7:

```
var_dump(version_compare(PHP_VERSION, '7.0.0') >= 0);
// bool(true)
var_dump(version_compare(PHP_VERSION, '7.0.0', '>='));
// bool(true)
```

- `getlastmod()` — возвращает время последнего изменения сценария:

```
echo date('d.m.y', getlastmod()); // Выведет: 06.01.18
```

- `get_current_user()` — позволяет узнать имя пользователя, являющегося владельцем запущенного сценария:

```
echo get_current_user();
```

5.21.3. Изменение значения директив во время выполнения сценария

С помощью функции `ini_set()` можно изменить значение какой-либо директивы из файла php.ini на время выполнения сценария. Формат функции:

```
ini_set(<Директива>, <Новое значение>)
```

С помощью функции `ini_get()` можно посмотреть текущее значение какой-либо директивы. Формат функции:

```
ini_get(<Директива>)
```

Пример:

```
echo ini_get('default_charset'); // Выведет: UTF-8
```

Функция `ini_get_all()` возвращает массив значений всех директив:

```
echo '<pre>';
print_r(ini_get_all());
echo '</pre>';
```

Фрагмент результата в окне Web-браузера:

```
...
    [date.timezone] => Array
        (
            [global_value] => Europe/Moscow
            [local_value] => Europe/Moscow
            [access] => 7
        )

    [default_charset] => Array
        (
            [global_value] => UTF-8
            [local_value] => UTF-8
            [access] => 7
        )
...
```

Установить значение директивы с помощью функции `ini_set()` можно не всегда. Опция `access`, возвращаемая функцией `ini_get_all()`, позволяет определить, можно ли изменить значение директивы. Она может принимать следующие значения:

- `4` — директива может быть изменена в файле php.ini или httpd.conf;
- `6` — директива может быть изменена в файле php.ini, .htaccess или httpd.conf;
- `7` — директива может быть изменена где угодно.

Посмотреть текущее значение какой-либо директивы и определить, откуда ее можно изменить, позволяет скрипт, приведенный в листинге 5.164.

Листинг 5.164. Получение значения директивы

```
<form action="<?=$_SERVER['SCRIPT_NAME']?>">
<input type="text" name="name_ini">
<input type="submit" value="Определить">
</form>
<?php
if (isset($_GET['name_ini'])) {
   $ini = $_GET['name_ini'];
   $arr = ini_get_all();
```

```
   if (!isset($arr[$ini])) {
      exit('Директива не найдена');
   }
   echo 'Директива <b>' . htmlspecialchars($ini) . '</b>';
   echo '<br>Глобальное значение: ';
   echo htmlspecialchars($arr[$ini]['global_value']);
   echo '<br>Локальное значение: ';
   echo htmlspecialchars($arr[$ini]['local_value']);
   echo '<br>Изменить можно ';
   switch ($arr[$ini]['access']) {
      case 4: echo 'в php.ini или httpd.conf'; break;
      case 6: echo 'в php.ini, .htaccess или httpd.conf'; break;
      case 7: echo 'где угодно'; break;
   }
}
```

Для изменения директив PHP из файла .htaccess или httpd.conf используются две директивы: `php_value` и `php_flag`. Директива `php_flag` служит для установки логических значений директив, а `php_value` — для строковых и числовых значений:

```
php_value <Директива> <Значение>
php_flag <Директива> On | Off
```

Пример:

```
php_value error_log "php_error_log"
php_flag log_errors On
```

5.21.4. Выполнение команд, содержащихся в строке

С помощью функции `eval()` можно выполнить строку как PHP-код (листинг 5.165).

Листинг 5.165. Выполнение команд, содержащихся в строке

```
<?php
$code =<<<'LABEL'
for ($i = 1; $i <= 5; $i++) {
   echo "Строка{$i} ";
}
LABEL;
try {
   eval($code);
} catch (\ParseError $e) {
   echo "Ошибка: $e";
}
```

Результат выполнения:

```
Строка1 Строка2 Строка3 Строка4 Строка5
```

Если код содержит синтаксические ошибки, то генерируется исключение `ParseError`, которое можно обработать с помощью инструкции `try`.

ГЛАВА 6



Основы MySQL. Работаем с базами данных

6.1. Основные понятия

MySQL — это система управления реляционными базами данных. Сервер MySQL позволяет эффективно работать с данными и обеспечивает быстрый доступ к информации одновременно нескольким пользователям. При этом доступ к данным предоставляется только пользователям, имеющим на это право.

Что же такое база данных? *Реляционная база данных* — это совокупность двумерных таблиц, связанных отношениями друг с другом. Каждая *таблица* содержит совокупность *записей*. В свою очередь запись — это набор *полей*, содержащих связанную информацию. Любое поле в базе данных имеет имя и определенный *тип*. Имя таблицы должно быть уникальным в пределах базы данных. В свою очередь имя поля должно быть уникальным в пределах таблицы.

Для выборки записей из базы данных разработан специализированный язык — SQL (Structured Query Language, структурированный язык запросов). С помощью этого языка можно создавать базы данных и таблицы, добавлять, изменять и удалять данные, получать данные по запросу. Но прежде чем изучать SQL, разберемся с тем, как создаются реляционные базы данных.

6.2. Нормализация базы данных

Для начала рассмотрим таблицу заказов (табл. 6.1).

Как видно из этой таблицы, господин Иванов Иван Иванович неоднократно делал покупки. И каждый раз в таблицу заказов добавлялись его адрес, город и телефон. А теперь представьте себе ситуацию, когда господин Иванов Иван Иванович сменил номер телефона, — каждую запись о его покупке пришлось бы изменить. Кроме того, напрасно тратится пространство на жестком диске.

По этим причинам имеет смысл вынести данные о клиенте в отдельную таблицу (табл. 6.2).

Таблица 6.1. *Заказы*

Name	Address	City	Phone	Product	Date_order	Price	Quantity	Total_price
Иванов Иван Иванович	Седова, 7	Санкт-Петербург	125-14-45	HDD	2017-10-20	3400	1	3400
Петров Сергей Николаевич	Невский, 88	Санкт-Петербург	312-12-51	Тюнер	2017-10-20	3100	1	3100
Иванов Иван Иванович	Седова, 7	Санкт-Петербург	125-14-45	Монитор	2017-10-25	7200	1	7200
Иванов Иван Иванович	Седова, 7	Санкт-Петербург	125-14-45	Тюнер	2017-10-30	3100	1	3100
Иванов Иван Иванович	Седова, 7	Санкт-Петербург	125-14-45	Ручка	2017-10-31	10	10	100
Иванов Иван Иванович	Седова, 7	Санкт-Петербург	125-14-45	Сканер	2017-10-31	6000	1	6000

Таблица 6.2. *Данные о клиентах*

id_Customer	Name	Address	City	Phone
1	Иванов Иван Иванович	Седова, 7	Санкт-Петербург	125-14-45
2	Петров Сергей Николаевич	Невский, 88	Санкт-Петербург	312-12-51

Теперь наша первоначальная таблица заказов (см. табл. 6.1) примет вид табл. 6.3.

Таблица 6.3. *Заказы*

id_Customer	Product	Date_order	Price	Quantity	Total_price
1	HDD	2017-10-20	3400	1	3400
2	Тюнер	2017-10-20	3100	1	3100
1	Монитор	2017-10-25	7200	1	7200
1	Тюнер	2017-10-30	3100	1	3100
1	Ручка	2017-10-31	10	10	100
1	Сканер	2017-10-31	6000	1	6000

Поле `id_Customer` в табл. 6.2 называется *первичным ключом* и содержит только уникальные записи, т. е. однозначно определяет строку в таблице. Поле `id_Customer` в табл. 6.3 называется *внешним ключом* и может содержать повторяющиеся записи.

Название города также можно вынести в отдельную таблицу (табл. 6.4).

Таблица 6.4. *Названия городов*

id_City	City
1	Санкт-Петербург
2	Москва

В итоге таблица данных о клиентах (см. табл. 6.2) примет вид табл. 6.5.

Таблица 6.5. *Данные о клиентах*

id_Customer	Name	Address	id_City	Phone
1	Иванов Иван Иванович	Седова, 7	1	125-14-45
2	Петров Сергей Николаевич	Невский, 88	1	312-12-51

Теперь то же самое можно сделать с названиями товаров (табл. 6.6).

Таблица 6.6. *Информация о товарах*

id_Product	Product	Price
1	HDD	3400
2	Тюнер	3100
3	Монитор	7200
4	Ручка	10
5	Сканер	6000

И таблица заказов еще уменьшится (табл. 6.7).

Таблица 6.7. *Заказы*

id_Customer	id_Product	Date_order	Quantity	Total_price
1	1	2017-10-20	1	3400
2	2	2017-10-20	1	3100
1	3	2017-10-25	1	7200
1	2	2017-10-30	1	3100
1	4	2017-10-31	10	100
1	5	2017-10-31	1	6000

Но и это еще не все. Создадим теперь табл. 6.8, содержащую элементы заказа.

Таблица 6.8. *Элементы заказа*

id_Order	id_Product	Quantity
1	1	1
2	2	1
3	3	1
4	2	1
5	4	10
5	5	1

В итоге табл. 6.1 примет вид табл. 6.9.

Таблица 6.9. *Таблица заказов после нормализации*

id_Order	id_Customer	Date_order	Total_price
1	1	2017-10-20	3400
2	2	2017-10-20	3100
3	1	2017-10-25	7200
4	1	2017-10-30	3100
5	1	2017-10-31	6100

Такой процесс оптимизации базы данных называется *нормализацией*.

Обратите внимание: в табл. 6.8 первичный ключ является составным (поля `id_Order` и `id_Product`).

На первый взгляд может показаться, что работать с такой базой данных проблематично. Но это не так. При изменении адреса или телефона покупателя достаточно поменять эти данные только в одной таблице. А отсутствие повторяющихся записей позволит снизить размер базы данных. О том, как получить данные сразу из нескольких таблиц, мы узнаем при изучении языка SQL. Но вначале следует рассмотреть типы данных, которые могут храниться в полях таблицы.

6.3. Типы данных полей

При создании любой таблицы необходимо принимать решение, какой тип данных будет содержать поле, т. к. в отличие, скажем, от массивов в PHP, в базе данных поле может содержать данные только одного типа. Для хранения разных типов данных требуется различный объем памяти. Следует выбирать тип данных, который требует меньшего объема памяти.

Типы данных делятся на числовые, строковые (в которых также можно запоминать бинарные данные) и типы для хранения даты и времени.

6.3.1. Числовые типы

Для хранения чисел используются поля следующих типов:

- `BIT [(<от 1 до 64>)]` — битовое значение;
- `TINYINT [(<Длина в символах>)]` — целые числа от `-128` до `127` или от `0` до `255`. Занимает 1 байт;
- `BOOL` или `BOOLEAN` — либо `0`, либо `1`. Синоним для `TINYINT(1)`. Занимает 1 байт;
- `SMALLINT [(<Длина в символах>)]` — целые числа от `-32 768` до `32 767` или от `0` до `65 535`. Занимает 2 байта;
- `MEDIUMINT [(<Длина в символах>)]` — целые числа от `-8 388 608` до `8 388 607` или от `0` до `16 777 215`. Занимает 3 байта;
- `INT [(<Длина в символах>)]` — целые 4-байтовые числа;
- `INTEGER [(<Длина в символах>)]` — синоним для `INT`;
- `BIGINT [(<Длина в символах>)]` — целые 8-байтовые числа;
- `SERIAL` — короткая запись для следующего объявления:

  `BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE`
- `FLOAT [(<Длина в символах>, <Число знаков после запятой>)]` — вещественные числа с диапазоном от `±1.175494351E-38` до `±3.402823466E+38`. Занимает 4 байта;
- `DOUBLE [(<Длина в символах>, <Число знаков после запятой>)]` — вещественные числа двойной точности. Занимает 8 байтов;
- `REAL` — синоним для `DOUBLE` или `FLOAT` (в режиме `REAL_AS_FLOAT`);
- `DECIMAL [(<Длина в символах>[, <Число знаков после запятой>])]` — дробные числа, обеспечивающие повышенную точность;
- `NUMERIC` — синоним для `DECIMAL`.

Если после типа указано слово `UNSIGNED`, значит, поле может содержать только числа без знака. Если указаны слова `UNSIGNED ZEROFILL`, то пустые символы будут заполняться нулями.

6.3.2. Строковые типы

Для хранения текста и бинарных данных можно использовать следующие типы:

- `CHAR (<Длина строки в символах>)` — строки фиксированной длины до 255 символов. Строки будут дополняться пробелами до максимальной длины, независимо от размеров строки;
- `VARCHAR (<Длина строки в символах>)` — строки переменной длины до 65 535 символов;

- `TINYTEXT` — строка до 255 символов;
- `TEXT` — строка до 65 535 символов;
- `MEDIUMTEXT` — строка до 16 777 215 символов;
- `LONGTEXT` — строка до 4 294 967 295 символов.

При поиске в текстовых полях регистр символов не учитывается.

Бинарные типы:

- `TINYBLOB` — до 255 байтов;
- `BLOB` — до 65 535 байтов;
- `MEDIUMBLOB` — до 16 777 215 байтов;
- `LONGBLOB` — до 4 294 967 295 байтов;
- `BINARY (<Длина строки в байтах>)` — то же, что `CHAR`, но строки хранятся в бинарном виде;
- `VARBINARY (<Длина строки в байтах>)` — то же, что `VARCHAR`, но строки хранятся в бинарном виде.

При поиске в бинарных полях учитывается регистр символов.

Перечисления и множества:

- `SET ('Значение1', 'Значение2', ...)` — поле может содержать несколько значений из перечисленных. Может быть указано до 64 значений;
- `ENUM ('Значение1', 'Значение2', ...)` — поле может содержать лишь одно из перечисленных значений или `NULL`. Может быть указано до 65 535 значений.

6.3.3. Дата и время

Календарные типы:

- `DATE` — дата в формате `ГГГГ-ММ-ДД`;
- `TIME` — время в формате `ЧЧ:ММ:СС`;
- `DATETIME` — дата и время в формате `ГГГГ-ММ-ДД ЧЧ:ММ:СС`;
- `YEAR [(4)]` — год в четырехсимвольном формате;
- `TIMESTAMP [(<Тип>)]` — дата и время в формате `timestamp` (хранится в виде числа секунд): от `'1970-01-01 00:00:00' UTC` до `'2038-01-19 03:14:07' UTC`.

6.4. Основы языка SQL

Для выборки записей из базы данных разработан специализированный язык — SQL (Structured Query Language, структурированный язык запросов). С помощью этого языка можно создавать базы данных и таблицы, добавлять, изменять и удалять данные, получать данные по запросу. В настоящее время существует множество разновидностей языка SQL. В этой главе книги мы будем изучать SQL примени-

тельно к базам данных MySQL. Обратите внимание: некоторые SQL-команды работают только в MySQL.

Команды языка SQL нечувствительны к регистру, но в книге они набраны прописными буквами.

6.4.1. Создание базы данных

Для создания базы данных используется команда:

```
CREATE DATABASE [IF NOT EXISTS] <Имя базы данных>;
```

Пример:

```
CREATE DATABASE `tests`;
```

При создании базы данных можно сразу выбрать кодировку:

```
CREATE DATABASE `tests` DEFAULT CHARACTER SET cp1251
   COLLATE cp1251_general_ci;
```

или

```
CREATE DATABASE `tests` DEFAULT CHARACTER SET utf8
   COLLATE utf8_general_ci;
```

> ***Примечание***
>
> Название базы данных заключают в обратные кавычки `` `tests` ``.

Если база данных с указанным нами именем уже существует, мы получим сообщение об ошибке. В этом случае обезопаситься можно, предусмотрев следующую команду:

```
CREATE DATABASE IF NOT EXISTS `tests` DEFAULT CHARACTER SET cp1251
   COLLATE cp1251_general_ci;
```

Для тестирования команд SQL можно воспользоваться программой phpMyAdmin, которая должна быть доступна по адресу **http://localhost/phpmyadmin/**. Не забудьте предварительно запустить серверы Apache и MySQL.

Итак, открываем программу phpMyAdmin. В правой части переходим на вкладку **SQL** и в текстовом поле набираем команду:

```
CREATE DATABASE `tests` DEFAULT CHARACTER SET utf8
   COLLATE utf8_general_ci;
```

Нажимаем кнопку **Вперед**. Появится сообщение: **MySQL вернула пустой результат (т. е. ноль строк). (Запрос занял 0,0000 сек.)**. Это значит, что команда создания базы данных была успешно выполнена, но, поскольку она не выполняет выборку данных, серверу нечего нам вывести.

При этом новая база данных появится в иерархическом списке на левой панели. Выбираем из этого списка пункт **tests** — в правой части окна отобразится содержимое базы данных `tests`, точнее сказать, надпись: **Таблиц в базе данных не обнаружено**, т. к. таблицы мы еще не создавали.

Среди всех баз данных может быть выбрана одна текущая, к которой направляются все команды SQL. Выбирается текущая база данных с помощью команды SQL:

```
USE <База данных>;
```

Например, только что созданную базу данных `tests` можно выбрать SQL-командой

```
USE `tests`;
```

В верхней части страницы расположены вкладки: **Структура**, **SQL**, **Поиск**, **Запрос по шаблону**, **Экспорт**, **Импорт**, **Операции** и др. Если в окне не будет хватать места для вывода всех этих вкладок, то в верхней части окна появится кнопка **Еще**, при нажатии на которую откроется меню со всеми не представленными на экране вкладками.

Далее нас будет интересовать вкладка **SQL**. Все дальнейшие SQL-запросы к базе данных мы будем набирать именно здесь. Итак, вначале выбираем базу данных из списка на левой панели, а затем выполняем SQL-запросы на вкладке **SQL**, расположенной на правой панели. Над правой панелью должна быть надпись: **База данных: tests**.

6.4.2. Создание пользователя базы данных

После создания базы данных необходимо создать пользователя базы данных и назначить ему полномочия. *Полномочия* (или *привилегии*) — это права определенного пользователя выполнять заданные действия над определенным объектом. Пользователь должен обладать наименьшим набором привилегий, необходимых для выполнения конкретных задач.

Создание и назначение полномочий осуществляются SQL-командой:

```
GRANT <Привилегии> [<Столбцы>]
ON <База данных>.<Таблица>
TO <Имя пользователя> [IDENTIFIED BY '<Пароль>']
[WITH GRANT OPTION];
```

В параметре `<Привилегии>` могут быть указаны через запятую следующие полномочия:

- `ALL` или `ALL PRIVILEGES` — все полномочия, кроме `GRANT OPTION` и `PROXY`;
- `USAGE` — без всех полномочий;
- `SELECT` — возможность выбирать записи в таблицах;
- `INSERT` — право вставлять новые записи в таблицы;
- `UPDATE` — полномочия изменять значения в существующих полях таблиц;
- `DELETE` — разрешение удалять записи;
- `FILE` — возможность сохранять данные из таблиц в файл и, наоборот, восстанавливать их из файла;

- CREATE — право создавать новые базы данных или таблицы. Если в команде GRANT указана определенная база данных или таблица, то пользователь может создавать только указанную базу данных или таблицу;
- CREATE TEMPORARY TABLES — разрешение создавать временные таблицы;
- CREATE USER — разрешение создавать, править и удалять пользователей;
- ALTER — полномочия изменять структуру существующих таблиц;
- INDEX — право создавать и удалять индексы определенных таблиц;
- DROP — возможность удаления базы данных или таблицы;
- PROCESS — разрешение просматривать процессы на сервере;
- RELOAD — возможность перезагружать таблицы полномочий;
- SHUTDOWN — право останавливать сервер MySQL;
- SHOW DATABASES — разрешение на просмотр списка всех баз данных на сервере.

В необязательном параметре <Столбцы> может быть указан список имен столбцов, разделенных запятыми, к которым применяются привилегии.

В параметре <База данных>.<Таблица> может быть указано:

- *.* или * — полномочия предоставляются для всех баз данных в целом;
- <Имя базы данных>.* — полномочия для всех таблиц указанной базы данных;
- <Имя базы данных>.<Имя таблицы> — привилегии относятся только к указанной таблице в указанной базе данных. Если дополнительно задан параметр <Столбцы>, то полномочия назначаются для указанных столбцов.

В параметре <Имя пользователя> указывается имя пользователя (например, den) или Имя_пользователя@Имя_хоста (например, den@localhost). Новому пользователю можно назначить пароль.

Если указана опция WITH GRANT OPTION, то пользователь может предоставлять свои полномочия другим.

Создадим нового пользователя с именем den и назначим ему ограниченные привилегии. Для этого на вкладке **SQL** набираем следующую команду:

```
GRANT select, insert, update, delete, index, alter, create, drop
ON `tests`.*
TO den@localhost IDENTIFIED BY '123';
```

и нажимаем кнопку **Вперед**. В итоге отобразится надпись: **MySQL вернула пустой результат (т. е. ноль строк). (Запрос занял 0,0000 сек.)**.

После создания пользователя или изменения привилегий необходимо перезагрузить привилегии с помощью SQL-команды:

```
FLUSH PRIVILEGES;
```

Для лишения пользователя полномочий служит команда SQL:

```
REVOKE <Привилегии> [<Столбцы>]
ON <База данных>.<Таблица>
FROM <Имя пользователя>;
```

Если полномочия были предоставлены опцией `WITH GRANT OPTION`, то удалить их можно с помощью команды SQL:

```
REVOKE GRANT OPTION
ON <База данных>.<Таблица>
FROM <Имя пользователя>;
```

Для просмотра прав пользователя предназначена команда SQL:

```
SHOW GRANTS FOR '<Имя пользователя>'@'<Хост>';
```

Для примера выведем полномочия созданного пользователя `den`:

```
SHOW GRANTS FOR 'den'@'localhost';
```

Для удаления пользователя используется SQL-команда:

```
DROP USER [IF EXISTS] '<Имя пользователя>'@'<Хост>';
```

6.4.3. Создание таблицы

Создать таблицу в базе данных позволяет SQL-команда:

```
CREATE TABLE [IF NOT EXISTS] <Имя таблицы> (
<Имя поля1> <Тип данных> [<Опции>],
<Имя поля2> <Тип данных> [<Опции>],
 ...
) [<Дополнительные опции>];
```

При попытке создать таблицу с именем, совпадающим с именем уже существующей в базе таблицы, будет сгенерирована ошибка. Чтобы исключить ее, следует вставить в команду создания таблицы слова `IF NOT EXISTS`.

В параметре `<Опции>` могут быть указаны следующие значения:

- `NOT NULL` — означает, что поле обязательно должно иметь значение при вставке новой записи в таблицу (если не задано значение по умолчанию). Если опция не указана, то поле может быть пустым;
- `PRIMARY KEY` — указывает, что поле является первичным ключом таблицы. Записи в таком поле должны быть уникальными. Опция также может быть указана после перечисления всех полей;
- `AUTO_INCREMENT` — указывает, что поле является счетчиком: если при вставке новой записи указать `NULL`, то MySQL автоматически генерирует значение, на единицу большее максимального значения, уже существующего в поле. В таблице может быть только одно поле с этой опцией;
- `DEFAULT` — задает для поля значение по умолчанию, которое будет использовано, если при вставке записи для этого поля не было явно указано значение;
- `CHARACTER SET` — определяет кодировку текстового поля;
- `COLLATE` — задает способ сравнения для текстового поля.

Возможные значения параметра `<Дополнительные опции>`:

- `ENGINE` — тип таблицы (например, `MyISAM` или `InnoDB`);

> **ПРИМЕЧАНИЕ**
>
> На практике обычно используются два типа таблиц: `MyISAM` и `InnoDB`. По умолчанию используется тип `InnoDB`. В отличие от типа `MyISAM`, таблицы типа `InnoDB` более устойчивы к сбоям, поддерживают транзакции и внешние ключи, но работают несколько медленнее.

- `[DEFAULT] CHARSET` — кодировка (например, `utf8`);
- `COLLATE` — задает способ сравнения;
- `AUTO_INCREMENT` — начальное значение для автоматической генерации значения поля.

Для вывода всех типов таблиц, поддерживаемых текущей версией MySQL, предназначена SQL-команда:

```
SHOW ENGINES;
```

Для вывода всех кодировок применяется SQL-команда:

```
SHOW CHARACTER SET;
```

Чтобы получить список всех типов сортировки, можно воспользоваться SQL-командой:

```
SHOW COLLATION;
```

Создадим таблицы из нашего первоначально рассмотренного примера. Для этого в левой части окна программы из списка выбираем базу **tests**, а в правой — вкладку **SQL**.

В текстовом поле набираем следующие команды:

```
CREATE TABLE `Cities` (
   `id_City` INT NOT NULL AUTO_INCREMENT,
   `City` VARCHAR(255) NOT NULL,
   PRIMARY KEY  (`id_City`)
) ENGINE=MyISAM CHARSET=utf8 COLLATE utf8_general_ci;

CREATE TABLE `Customers` (
   `id_Customer` INT NOT NULL AUTO_INCREMENT,
   `Name` VARCHAR(255) NOT NULL,
   `Address` VARCHAR(255) NOT NULL,
   `id_City` INT NOT NULL,
   `Phone` VARCHAR(255),
   PRIMARY KEY  (`id_Customer`)
) ENGINE=MyISAM CHARSET=utf8 COLLATE utf8_general_ci;

CREATE TABLE `Products` (
   `id_Product` INT NOT NULL AUTO_INCREMENT,
```

```
   `Product` VARCHAR(255) NOT NULL,
   `Price` INT NOT NULL,
   PRIMARY KEY  (`id_Product`)
) ENGINE=MyISAM CHARSET=utf8 COLLATE utf8_general_ci;

CREATE TABLE `Orders_Items` (
   `id_Order` INT NOT NULL,
   `id_Product` INT NOT NULL,
   `Quantity` INT UNSIGNED,
   PRIMARY KEY  (`id_Order`, `id_Product`)
) ENGINE=MyISAM CHARSET=utf8 COLLATE utf8_general_ci;

CREATE TABLE `Orders` (
   `id_Order` INT NOT NULL AUTO_INCREMENT,
   `id_Customer` INT NOT NULL,
   `Date_order` DATE,
   `Total_price` INT,
   PRIMARY KEY  (`id_Order`)
) ENGINE=MyISAM CHARSET=utf8 COLLATE utf8_general_ci;
```

Можно набрать все команды одновременно, а можно и по отдельности. Чтобы выполнить запрос, нажимаем кнопку **Вперед**. Все созданные таблицы отображаются слева под списком баз данных (пункт **Новая** ведет на страницу создания новой таблицы):

```
tests
   Новая
   cities
   customers
   orders
   orders_items
   products
```

Если таблицы не отобразились, то обновите страницу.

Если щелкнуть на названии таблицы, то справа отобразятся вкладки, с помощью которых можно увидеть содержимое таблицы, ее структуру и т. д.

Вывести все таблицы из указанной базы данных позволяет SQL-команда:

```
SHOW TABLES FROM <Имя базы данных>;
```

Для примера выведем все таблицы из базы данных `tests`:

```
SHOW TABLES FROM `tests`;
```

Чтобы отобразить структуру конкретной таблицы из указанной базы данных, можно воспользоваться командой SQL:

```
SHOW COLUMNS FROM <Таблица> FROM <Имя базы данных>;
```

Для примера выведем структуру таблицы `Cities` из базы данных `tests`:

```
SHOW COLUMNS FROM `Cities` FROM `tests`;
```

Для отображения структуры таблицы можно также воспользоваться операторами EXPLAIN или DESCRIBE (синоним EXPLAIN):

```
EXPLAIN | DESCRIBE <Имя таблицы>;
```

Пример:

```
EXPLAIN `Cities`;
```

6.4.4. Добавление данных в таблицу

Для добавления записей в таблицу используется SQL-команда:

```
INSERT INTO <Имя таблицы> [(<Поле1>, <Поле2>, ...)]
VALUES ('<Значение1>', '<Значение2>', ...);
INSERT INTO <Имя таблицы>
SET <Поле1>='<Значение1>', <Поле2>='<Значение2>', ...;
```

Например, добавить две записи в таблицу Cities можно одним из следующих способов:

```
INSERT INTO `Cities` (`id_City`, `City`)
VALUES (NULL, 'Санкт-Петербург');
INSERT INTO `Cities` (`id_City`, `City`)
VALUES (NULL, 'Москва');

INSERT INTO `Cities` (`City`)
VALUES ('Санкт-Петербург');
INSERT INTO `Cities` (`City`)
VALUES ('Москва');

INSERT INTO `Cities`
SET `id_City`=NULL, `City`='Санкт-Петербург';
INSERT INTO `Cities`
SET `id_City`=NULL, `City`='Москва';

INSERT INTO `Cities`
SET `City`='Санкт-Петербург';
INSERT INTO `Cities`
SET `City`='Москва';

INSERT INTO `Cities` VALUES
(NULL, 'Санкт-Петербург'),
(NULL, 'Москва');

INSERT INTO `Cities` VALUES (NULL, 'Санкт-Петербург');
INSERT INTO `Cities` VALUES (NULL, 'Москва');
```

Обратите внимание: для первого поля мы указали значение NULL, т. к. для этого поля установлена опция AUTO_INCREMENT, и MySQL автоматически вставит значение в поле.

Если название таблицы содержит пробел или совпадает с одним из ключевых слов MySQL, то название таблицы необходимо заключить в обратные кавычки.

Давайте теперь заполним наши созданные таблицы значениями. Для этого выполним следующие SQL-команды:

```
INSERT INTO `Cities` VALUES
(1, 'Санкт-Петербург'),
(2, 'Москва');

INSERT INTO `Customers` VALUES
(1, 'Иванов Иван Иванович', 'Седова, 7', 1, '125-14-45'),
(2, 'Петров Сергей Николаевич', 'Невский, 88', 1, '312-12-51');

INSERT INTO `Products` VALUES
(1, 'HDD', 3400),
(2, 'Тюнер', 3100),
(3, 'Монитор', 7200),
(4, 'Ручка', 10),
(5, 'Сканер', 6000);

INSERT INTO `Orders_Items` VALUES
(1, 1, 1),
(2, 2, 1),
(3, 3, 1),
(4, 2, 1),
(5, 4, 10),
(5, 5, 1);

INSERT INTO `Orders` VALUES
(1, 1, '2017-10-20', 3400),
(2, 2, '2017-10-20', 3100),
(3, 1, '2017-10-25', 7200),
(4, 1, '2017-10-30', 3100),
(5, 1, '2017-10-31', 6100);
```

Обратите внимание: числа в кавычки не заключаются. А чтобы сохранить целостность базы данных, индексы указываются явным образом.

Если предпринимается попытка вставить запись, а в таблице уже есть запись с таким же значением первичного ключа (или значение индекса `UNIQUE` не уникально), то такая SQL-команда приводит к ошибке. Если необходимо, чтобы подобные неуникальные записи обновлялись без вывода сообщения об ошибке, можно использовать следующую SQL-команду:

```
REPLACE [INTO] <Имя таблицы> [(<Поле1>, <Поле2>, ...)]
VALUES ('<Значение1>', '<Значение2>', ...);
```

В качестве примера изменим номер телефона господина Иванова:

```
REPLACE `Customers` VALUES
(1, 'Иванов Иван Иванович', 'Седова, 7', 1, '125-14-47');
```

Если передать уникальное значение, то SQL-команда REPLACE аналогична команде INSERT. Например, следующая SQL-команда добавит нового покупателя:

```
REPLACE `Customers` VALUES
(NULL, 'Сидоров Олег Николаевич', 'Передовиков, 12', 1, '529-15-63');
```

6.4.5. Обновление записей

Обновление записи осуществляется SQL-командой:

```
UPDATE <Имя таблицы>
SET <Поле1>='<Значение1>', <Поле2>='<Значение2>', ...
WHERE <Условие>;
```

> ***Внимание!***
> Если не указано <Условие>, то будут обновлены все записи в таблице.

В параметре <Условие> могут быть указаны следующие операторы:

- = — проверка на равенство;
- > — больше;
- < — меньше;
- >= — больше или равно;
- <= — меньше или равно;
- != или <> — не равно;
- IS NOT NULL — проверка на наличие значения;
- IS NULL — проверка поля на отсутствие значения;
- BETWEEN <Начало> AND <Конец> — проверяет, является ли значение большим или равным значению <Начало> и меньшим или равным значению <Конец>. Например: `id_Customer` BETWEEN 1 AND 3;
- IN — содержится в определенном наборе. Например: `Product` IN ('Монитор', 'HDD');
- NOT IN — не содержится в определенном наборе. Например: `Product` NOT IN ('Монитор', 'HDD');
- LIKE — соответствие шаблону SQL. Например: `Product` LIKE 'P%';
- NOT LIKE — несоответствие шаблону SQL.

В шаблоне SQL могут использоваться следующие символы:

- % — любое число символов;
- _ — любой одиночный символ.

Можно проверять сразу несколько условий, соединив их логическими операциями:

- AND или && — логическое И;
- OR или || — логическое ИЛИ;

- XOR — логическое исключающее ИЛИ;
- NOT или ! — логическое отрицание.

Если название таблицы содержит пробел или совпадает с одним из ключевых слов MySQL, то название таблицы необходимо заключить в обратные кавычки. Для примера изменим телефон одного из клиентов, например Иванова:

```
UPDATE `Customers` SET `Phone`='125-14-46' WHERE `id_Customer`=1;
```

Господин Иванов у нас числится под номером 1 в таблице Customers. Это условие мы и указали.

6.4.6. Удаление записей из таблицы

Удаление записи осуществляется SQL-командой:

```
DELETE FROM <Имя таблицы> WHERE <Условие> [ LIMIT <Число> ];
```

> ***Внимание!***
> Если <Условие> не указано, то будут удалены все записи из таблицы.

С помощью конструкции LIMIT можно ограничить максимальное число удаляемых записей. В качестве примера удалим клиента по фамилии Сидоров:

```
DELETE FROM `Customers` WHERE `Name` LIKE 'Сидоров %' LIMIT 1;
```

В этом примере мы удалим первого клиента с фамилией Сидоров. Клиентов с одинаковыми фамилиями может быть очень много, и таким запросом мы можем удалить не того клиента. В нашем примере все будет работать правильно, т. к. клиент всего один. На практике нужно знать уникальный номер клиента и именно его указывать в качестве условия:

```
DELETE FROM `Customers` WHERE `id_Customer`=3;
```

Для очистки определенной таблицы служит SQL-команда:

```
TRUNCATE TABLE <Имя таблицы>;
```

Частое обновление и удаление записей приводит к фрагментации таблицы. Чтобы освободить неиспользуемое свободное пространство, можно воспользоваться SQL-командой:

```
OPTIMIZE TABLE <Имя таблицы>;
```

Если таблица была повреждена, то восстановить ее позволяет SQL-команда:

```
REPAIR TABLE <Имя таблицы>;
```

6.4.7. Изменение структуры таблицы

В ряде случаев нужно изменить структуру уже созданной таблицы. Для этого используется SQL-команда:

```
ALTER TABLE <Имя таблицы>
<Преобразование>;
```

В параметре <Преобразование> могут быть указаны следующие инструкции:

- RENAME <Новое имя таблицы> — переименовывает таблицу;
- ADD <Имя нового поля> <Тип данных> [FIRST | AFTER <Имя поля>] — добавляет в таблицу новое поле. Если указана опция FIRST, то поле будет добавлено в самое начало, а если AFTER <Имя поля> — то после указанного поля. По умолчанию новое поле вставляется в конец таблицы. Обратите внимание: в новом поле нужно задать значение по умолчанию, или значение NULL должно быть допустимым, т. к. в таблице уже есть записи;
- ADD PRIMARY KEY (<Имя поля>) — делает указанное поле первичным ключом;
- DROP PRIMARY KEY — удаляет первичный ключ;
- CHANGE <Имя поля> <Новое имя поля> <Новые параметры поля> — изменяет свойства поля. С помощью этой инструкции поле можно переименовать. Если этого не требуется, то <Новое имя поля> должно содержать то же имя, что и <Имя поля>;
- MODIFY <Имя поля> <Тип данных> — изменяет свойства поля;
- DROP <Имя поля> — удаляет поле.

Для примера изменим тип данных поля Address в таблице Customers:

```
ALTER TABLE `Customers`
CHANGE `Address` `Address` VARCHAR(200) NOT NULL;
```

6.4.8. Выбор записей

Выполнить запрос позволяет SQL-команда:

```
SELECT <Поле1>, <Поле2>, ...
FROM <Имя таблицы>
[ WHERE <Условие1> ]
[ GROUP BY <Имя поля1> ] [ HAVING <Условие2> ]
[ ORDER BY <Имя поля2> [DESC]]
[ LIMIT <Начало>, <Число записей> ]
```

SQL-команда SELECT ищет все записи в таблице <Имя таблицы>, которые удовлетворяют выражению <Условие1>. Если конструкция WHERE <Условие1> опущена, то будут возвращены все записи из таблицы <Имя таблицы>. Вместо перечисления полей можно указать символ *. В этом случае будут возвращены все поля.

Найденные записи при указанной конструкции ORDER BY <Имя поля2> сортируются по возрастанию. Если в конце указано слово DESC, то записи будут отсортированы в обратном порядке.

Для начала выберем все записи из таблицы Cities, но лишь из одного поля:

```
SELECT `City` FROM `Cities`;
```

В результате будут возвращены только названия городов:

```
Санкт-Петербург
Москва
```

Если вместо названия поля указать символ `*`, то будут возвращены все поля:

```
SELECT * FROM `Cities`;
```

Этот запрос вернет:

```
1 Санкт-Петербург
2 Москва
```

Выведем названия городов по алфавиту:

```
SELECT * FROM `Cities` ORDER BY `City`;
```

Теперь названия будут отсортированы:

```
2 Москва
1 Санкт-Петербург
```

Выведем только город с индексом `2`:

```
SELECT * FROM `Cities` WHERE `id_City`=2;
```

В результате мы получим один город:

```
2 Москва
```

Если требуется, чтобы при поиске выдавались не все найденные записи, а лишь их часть, то нужно использовать параметр `LIMIT`. Этот параметр удобен при выводе большого числа записей. Например, есть каталог из 2 000 записей. Вместо того чтобы выводить его за один раз, можно выводить его частями — скажем, по 25 записей за раз. В параметре `LIMIT` задаются два значения: `<Начало>` и `<Число записей>`:

```
SELECT * FROM `Cities` ORDER BY `City` LIMIT 0, 25;
```

Обратите внимание, что первая запись имеет индекс `0`. Если в таблице `Cities` было бы более 25 записей, то мы бы получили только первые 25.

Выберем следующие 25 записей:

```
SELECT * FROM `Cities` ORDER BY `City` LIMIT 25, 25;
```

Кроме того, команда `SELECT` позволяет использовать следующие функции, называемые *агрегатными функциями*:

- `COUNT(<Поле>)` — число непустых (т. е. не имеющих значение `NULL`) записей в указанном поле;
- `MIN(<Поле>)` — минимальное значение в указанном поле;
- `MAX(<Поле>)` — максимальное значение в указанном поле;
- `SUM(<Поле>)` — сумма значений в указанном поле;
- `AVG(<Поле>)` — средняя величина значений в указанном поле.

Выведем общее количество заказов:

```
SELECT COUNT(`id_Order`) FROM `Orders`;
```

Этот SQL-запрос выведет: `5`.

Теперь найдем минимальную сумму заказа:

```
SELECT MIN(`Total_price`) FROM `Orders`;
```

В результате получим: 3100.

А теперь выясним максимальную сумму заказа:

```
SELECT MAX(`Total_price`) FROM `Orders`;
```

И получим ответ: 7200.

Для получения более подробной информации можно воспользоваться конструкцией GROUP BY. Например, так можно посмотреть среднюю сумму покупок каждого покупателя:

```
SELECT `id_Customer`, AVG(`Total_price`) AS s
FROM `Orders`
GROUP BY `id_Customer`
ORDER BY s;
```

> ***Внимание!***
>
> Мы используем псевдоним для имени поля и по нему сортируем записи от меньшего результата к большему.

Если нужно, например, выбрать клиентов, заказавших больше определенной суммы, то можно воспользоваться конструкцией HAVING. Она выполняет те же функции, что и конструкция WHERE, но только для конструкции GROUP BY:

```
SELECT `id_Customer`, SUM(`Total_price`) AS s
FROM `Orders`
GROUP BY `id_Customer`
HAVING s > 4000
ORDER BY s;
```

Можно в одном запросе использовать конструкции и WHERE, и HAVING. В этом случае сперва отбираются записи, указанные в конструкции WHERE, они группируются, и по ним вычисляются агрегатные функции, а затем из результата отбираются лишь те записи, которые удовлетворяют условию в конструкции HAVING.

6.4.9. Выбор записей из нескольких таблиц

SQL-команда SELECT позволяет выбирать записи сразу из нескольких таблиц одновременно. Для этого нужно указать все таблицы через запятую в конструкции FROM. В конструкции WHERE через запятую указываются пары полей, являющиеся связуемыми для таблиц. Причем в условии и перечне полей вначале указывается имя таблицы, а затем — через точку — имя поля.

Для примера выведем таблицу Customers, но вместо индекса города укажем его название:

```
SELECT `Customers`.`Name`, `Customers`.`Address`, `Cities`.`City`,
       `Customers`.`Phone`
FROM `Customers`, `Cities`
WHERE `Customers`.`id_City`=`Cities`.`id_City`;
```

В итоге мы получим табл. 6.10.

Таблица 6.10. *Данные о клиентах*

Name	Address	City	Phone
Иванов Иван Иванович	Седова, 7	Санкт-Петербург	125-14-46
Петров Сергей Николаевич	Невский, 88	Санкт-Петербург	312-12-51

Название таблицы можно заменить псевдонимом, который создается через ключевое слово `AS` после имени таблицы в конструкции `FROM`. Перепишем предыдущий пример с использованием псевдонимов:

```
SELECT `c`.`Name`, `c`.`Address`, `ct`.`City`, `c`.`Phone`
FROM `Customers` AS `c`, `Cities` AS `ct`
WHERE `c`.`id_City`=`ct`.`id_City`;
```

Результат будет таким же. Кроме того, если поля в таблицах имеют разные названия, то имя таблицы можно не указывать:

```
SELECT `Name`, `Address`, `City`, `Phone`
FROM `Customers` AS `c`, `Cities` AS `ct`
WHERE `c`.`id_City`=`ct`.`id_City`;
```

А теперь выведем нашу первоначальную таблицу (см. табл. 6.1):

```
SELECT `c`.`Name`, `c`.`Address`, `ct`.`City`, `c`.`Phone`,
       `p`.`Product`, `o`.`Date_order`, `p`.`Price`, `oi`.`Quantity`
FROM `Customers` AS `c`, `Cities` AS `ct`, `Products` AS `p`,
     `Orders` AS `o`, `Orders_Items` AS `oi`
WHERE `c`.`id_City`=`ct`.`id_City` AND
      `oi`.`id_Order`=`o`.`id_Order` AND
      `p`.`id_Product`=`oi`.`id_Product` AND
      `o`.`id_Customer`=`c`.`id_Customer`
ORDER BY `o`.`id_Order`, `p`.`Product`;
```

В итоге мы получим табл. 6.11.

Таблица 6.11. *Таблица заказов*

Name	Address	City	Phone	Product	Date_order	Price	Quantity
Иванов Иван Иванович	Седова, 7	Санкт-Петербург	125-14-46	HDD	2017-10-20	3400	1
Петров Сергей Николаевич	Невский, 88	Санкт-Петербург	312-12-51	Тюнер	2017-10-20	3100	1
Иванов Иван Иванович	Седова, 7	Санкт-Петербург	125-14-46	Монитор	2017-10-25	7200	1
Иванов Иван Иванович	Седова, 7	Санкт-Петербург	125-14-46	Тюнер	2017-10-30	3100	1

Таблица 6.11 *(окончание)*

Name	Address	City	Phone	Product	Date_order	Price	Quantity
Иванов Иван Иванович	Седова, 7	Санкт-Петербург	125-14-46	Ручка	2017-10-31	10	10
Иванов Иван Иванович	Седова, 7	Санкт-Петербург	125-14-46	Сканер	2017-10-31	6000	1

Эта таблица практически совпадает с табл. 6.1, но с двумя исключениями:

- нет поля `Total_price`. Получить это поле несложно — достаточно перемножить значения полей `Price` и `Quantity`;
- номер телефона господина Иванова изменился, т. к. мы его чуть раньше сами поменяли.

Связывать таблицы можно также с помощью оператора `JOIN`. Для примера выведем таблицу `Customers`, но вместо индекса города укажем его название:

```
SELECT `Customers`.`Name`, `Customers`.`Address`, `Cities`.`City`,
       `Customers`.`Phone`
FROM `Customers` JOIN `Cities`
ON `Customers`.`id_City`=`Cities`.`id_City`;
```

Отметим, что в этом случае вместо оператора `WHERE` мы используем `ON`.

Если необходимо указать дополнительное условие выборки, то это делают в инструкции `WHERE`.

Для примера выведем информацию о клиентах с фамилией Иванов:

```
SELECT `Customers`.`Name`, `Customers`.`Address`, `Cities`.`City`,
       `Customers`.`Phone`
FROM `Customers` JOIN `Cities`
ON `Customers`.`id_City`=`Cities`.`id_City`
WHERE `Customers`.`Name` LIKE 'Иванов %';
```

Если названия полей в таблицах одинаковые, то инструкцию `ON` можно заменить на `USING`:

```
SELECT `Customers`.`Name`, `Customers`.`Address`, `Cities`.`City`,
       `Customers`.`Phone`
FROM `Customers` JOIN `Cities` USING (`id_City`);
```

Оператор `JOIN` позволяет также объединить несколько таблиц. В качестве примера выведем нашу первоначальную таблицу (см. табл. 6.1):

```
SELECT `c`.`Name`, `c`.`Address`, `ct`.`City`, `c`.`Phone`,
       `p`.`Product`, `o`.`Date_order`, `p`.`Price`, `oi`.`Quantity`
FROM `Customers` AS `c` JOIN `Cities` AS `ct` JOIN `Products` AS `p`
JOIN `Orders` AS `o` JOIN `Orders_Items` AS `oi`
ON `c`.`id_City`=`ct`.`id_City` AND
   `oi`.`id_Order`=`o`.`id_Order` AND
```

```
   `p`.`id_Product`=`oi`.`id_Product` AND
   `o`.`id_Customer`=`c`.`id_Customer`
ORDER BY `o`.`id_Order`, `p`.`Product`;
```

> ***Примечание***
>
> Оператор `JOIN` имеет два синонима: `CROSS JOIN` и `INNER JOIN`.

Добавим нового клиента в таблицу `Customers` и выведем общее число заказов каждого клиента:

```
INSERT INTO `Customers` VALUES
(NULL, 'Сидоров Олег Николаевич', 'Передовиков, 12', 1, '529-15-63');
SELECT `Customers`.`Name`, COUNT(`Orders`.`id_Order`)
FROM `Customers` JOIN `Orders` USING (`id_Customer`)
GROUP BY `Orders`.`id_Customer`;
```

Получим следующий результат:

```
Иванов Иван Иванович       4
Петров Сергей Николаевич   1
```

Как видно из примера, этот запрос вывел только клиентов, сделавших хотя бы один заказ. Так как господин Сидоров не сделал ни одного заказа, то в таблице `Orders` отсутствует запись о нем. Чтобы получить всех клиентов, необходимо использовать *левостороннее объединение* с помощью инструкции `LEFT JOIN`:

```
<Таблица1> LEFT [OUTER] JOIN <Таблица2> ON
<Таблица1>.<Поле1>=<Таблица2>.<Поле2>
```

Ключевое слово `OUTER` необязательное — его поддержка оставлена для совместимости со стандартом SQL.

Если названия полей в таблицах одинаковые, то вместо инструкции `ON` можно использовать инструкцию `USING`:

```
<Таблица1> LEFT [OUTER] JOIN <Таблица2> USING (<Поле>)
```

При левостороннем объединении возвращаются записи, соответствующие условию `<Таблица1>.<Поле1>=<Таблица2>.<Поле2>`, а также записи из таблицы `<Таблица1>`, которым нет соответствия в таблице `<Таблица2>` (при этом поля из таблицы `<Таблица2>` будут иметь значение `NULL`).

Выведем общее число заказов каждого клиента с помощью левостороннего объединения:

```
SELECT `Customers`.`Name`, COUNT(`Orders`.`id_Order`) AS `total`
FROM `Customers` LEFT JOIN `Orders` USING (`id_Customer`)
GROUP BY `Orders`.`id_Customer`
ORDER BY `total` DESC;
```

Получим следующий результат:

```
Иванов Иван Иванович       4
Петров Сергей Николаевич   1
Сидоров Олег Николаевич    0
```

Кроме левостороннего, можно применить *правостороннее объединение* с помощью инструкции `RIGHT JOIN`:

```
<Таблица1> RIGHT [OUTER] JOIN <Таблица2> ON
<Таблица1>.<Поле1>=<Таблица2>.<Поле2>
```

Здесь также допустима конструкция с ключевым словом `USING`, если названия полей в обеих таблицах совпадают:

```
<Таблица1> RIGHT [OUTER] JOIN <Таблица2> USING (<Поле>)
```

При правостороннем объединении возвращаются записи, соответствующие условию `<Таблица1>.<Поле1>=<Таблица2>.<Поле2>`, а также записи из таблицы `<Таблица2>`, которым нет соответствия в таблице `<Таблица1>` (при этом поля из таблицы `<Таблица1>` будут иметь значение `NULL`).

Выведем общее число заказов каждого клиента с помощью правостороннего объединения:

```
SELECT `Customers`.`Name`, COUNT(`Orders`.`id_Order`) AS `total`
FROM `Orders` RIGHT JOIN `Customers` USING (`id_Customer`)
GROUP BY `Orders`.`id_Customer`
ORDER BY `total` DESC;
```

В этом примере мы просто поменяли местами таблицы в инструкции `FROM`:

```
FROM `Orders` RIGHT JOIN `Customers`
```

6.4.10. Индексы. Ускорение выполнения запросов

Для определения эффективности SQL-запроса предусмотрены операторы `EXPLAIN` и `DESCRIBE`. Они выполняют одну и ту же задачу и имеют следующий формат:

```
EXPLAIN | DESCRIBE <Имя таблицы>;
EXPLAIN | DESCRIBE <SQL-запрос>;
```

Первый вариант выведет структуру указанной таблицы, а второй вариант позволяет выяснить, каким образом выполняется запрос с помощью SQL-команды `SELECT`.

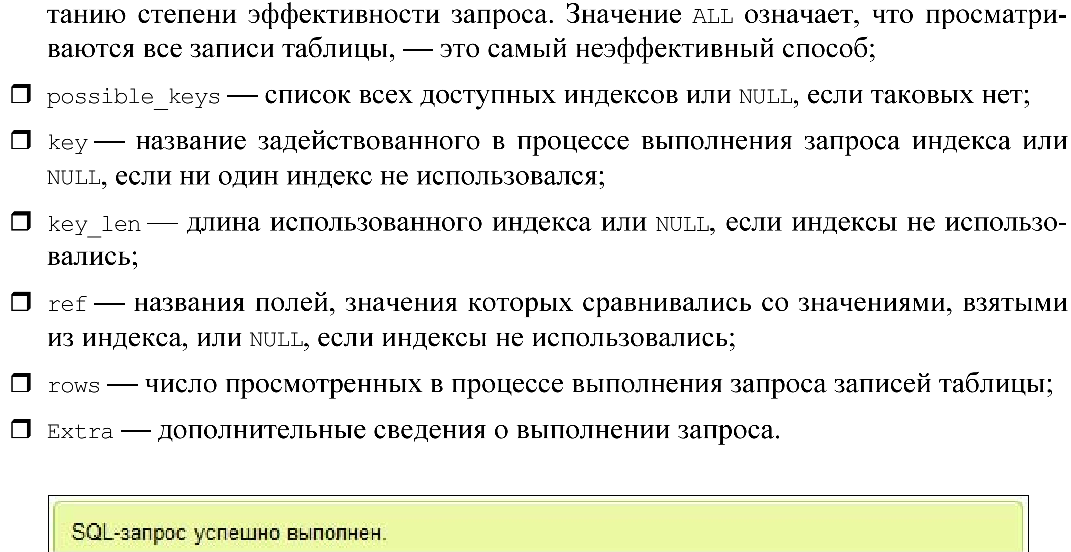
В качестве примера выведем результат поиска клиента по его полному имени:

```
EXPLAIN SELECT * FROM `Customers` WHERE `Name`='Иванов Иван Иванович';
```

Эта команда вернет результат, показанный на рис. 6.1.

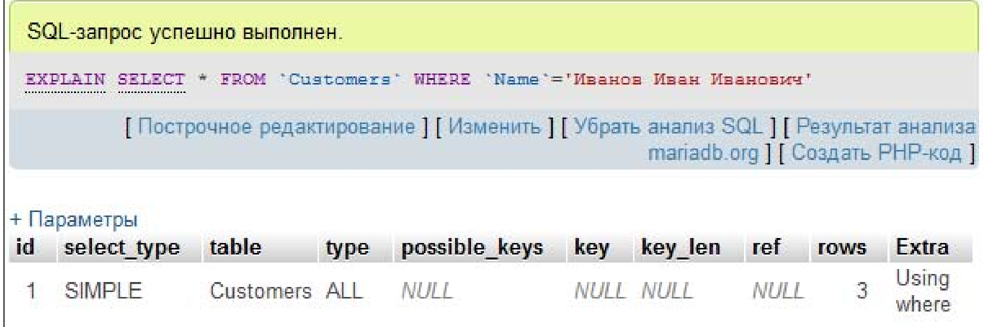
Рассмотрим результат выполнения команды `EXPLAIN` по столбцам:

- `id` — порядковый номер выполненного запроса;
- `select_type` — тип запроса. В нашем случае: `SIMPLE`, т. е. простой запрос;
- `table` — название таблицы или таблиц, к которым был выполнен запрос;
- `type` — эффективность выполнения запроса. Может принимать значения `ALL`, `index`, `range`, `index_subquery`, `unique_subquery`, `index_merge`, `ref_or_null`, `fulltext`, `ref`, `eq_ref`, `const` и `system`. Приведенные здесь значения расположены по возрас-



таниию степени эффективности запроса. Значение `ALL` означает, что просматриваются все записи таблицы, — это самый неэффективный способ;

- `possible_keys` — список всех доступных индексов или `NULL`, если таковых нет;
- `key` — название задействованного в процессе выполнения запроса индекса или `NULL`, если ни один индекс не использовался;
- `key_len` — длина использованного индекса или `NULL`, если индексы не использовались;
- `ref` — названия полей, значения которых сравнивались со значениями, взятыми из индекса, или `NULL`, если индексы не использовались;
- `rows` — число просмотренных в процессе выполнения запроса записей таблицы;
- `Extra` — дополнительные сведения о выполнении запроса.



id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	Customers	ALL	NULL	NULL	NULL	NULL	3	Using where

Рис. 6.1. Результат, возвращенный командой `EXPLAIN`

Как видно из приведенного примера, для выполнения запроса пришлось просматривать все записи таблицы `Customers`, т. к. записи в неиндексированных полях таблицы расположены в произвольном порядке.

Для ускорения выполнения запросов применяются *индексы* (*ключи*). Индексированные поля всегда поддерживаются в отсортированном состоянии, что позволяет быстро найти необходимую запись, не просматривая все записи. Неиндексированное поле можно сравнить с книгой без предметного указателя, а индексированное поле — с книгой, где он присутствует. Чтобы найти что-либо в первом случае, необходимо последовательно перелистывать страницы книги. Во втором случае достаточно отыскать нужное понятие по алфавиту в предметном указателе, а затем сразу перейти на указанную страницу.

Необходимо заметить, что применение индексов приводит к увеличению размера базы данных, а также к затратам времени на поддержание индекса в отсортированном состоянии при каждом добавлении данных. Поэтому индексировать следует поля, которые очень часто используются в запросах типа:

```
SELECT <Список полей> FROM <Таблица> WHERE <Поле>=<Значение>;
```

Существуют следующие виды индексов:

- первичный ключ;
- уникальный индекс;
- обычный индекс;
- индекс FULLTEXT.

Первичный ключ служит для однозначной идентификации каждой записи в таблице. Для создания индекса предусмотрено ключевое слово PRIMARY KEY. При создании таблицы ключевое слово можно указать после определения параметров поля:

```
CREATE TABLE `Cities` (
   `id_City` INT NOT NULL PRIMARY KEY AUTO_INCREMENT,
   `City` VARCHAR(255) NOT NULL
) ENGINE=MyISAM CHARSET=utf8 COLLATE utf8_general_ci;
```

или после указания всех полей:

```
CREATE TABLE `Cities` (
   `id_City` INT NOT NULL AUTO_INCREMENT,
   `City` VARCHAR(255) NOT NULL,
   PRIMARY KEY  (`id_City`)
) ENGINE=MyISAM CHARSET=utf8 COLLATE utf8_general_ci;
```

Вторым способом можно создать первичный ключ, состоящий из нескольких полей (нужно записать их в скобках через запятую):

```
PRIMARY KEY  (`id_Order`, `id_Product`)
```

Добавить первичный ключ в существующую таблицу позволяет SQL-команда:

```
ALTER TABLE <Таблица> ADD PRIMARY KEY (<Поле>);
```

Удалить первичный ключ позволяет SQL-команда (если для поля указана опция AUTO_INCREMENT, то вначале нужно удалить эту опцию):

```
ALTER TABLE <Таблица> DROP PRIMARY KEY;
```

В одной таблице не может быть более одного первичного ключа. А вот обычных и уникальных индексов в таблице может быть несколько. Создать индекс можно при определении структуры таблицы с помощью ключевых слов INDEX и KEY (UNIQUE INDEX и UNIQUE KEY для уникального индекса):

```
CREATE TABLE `Customers` (
   `id_Customer` INT NOT NULL AUTO_INCREMENT,
   `Name` VARCHAR(255) NOT NULL,
   `Address` VARCHAR(255) NOT NULL,
   `id_City` INT NOT NULL,
   `Phone` VARCHAR(255),
   PRIMARY KEY  (`id_Customer`),
   KEY MyIndex (`Name`)
) ENGINE=MyISAM CHARSET=utf8 COLLATE utf8_general_ci;
```

Индекс может иметь название. Но поскольку название индекса не указывается в SQL-запросе, то чаще всего названием индекса служит имя поля. Сервер MySQL самостоятельно решает, каким индексом лучше воспользоваться в каждой конкретной ситуации. Знать название индекса необходимо для его удаления из таблицы.

При индексировании текстовых полей следует указать число символов, подлежащих индексации:

```
KEY MyIndex (`Name` (10))
```

> ***Примечание***
>
> В большинстве случаев достаточно внести в индекс первые четыре или пять символов.

Создать обычный индекс позволяют SQL-команды:

```
CREATE INDEX <Имя индекса> ON <Таблица> (<Поле>(<Число символов>));
```

или

```
ALTER TABLE <Таблица>
ADD INDEX <Имя индекса> (<Поле>(<Число символов>));
```

Создать уникальный индекс можно с помощью SQL-команд:

```
CREATE UNIQUE INDEX <Имя индекса>
ON <Таблица> (<Поле>(<Число символов>));
```

или

```
ALTER TABLE <Таблица>
ADD UNIQUE INDEX <Имя индекса> (<Поле>(<Число символов>));
```

Удалить обычный и уникальный индексы позволяют SQL-команды:

```
DROP INDEX <Имя индекса> ON <Таблица>;
```

или

```
ALTER TABLE <Таблица> DROP INDEX <Имя индекса>;
```

В качестве примера создадим индекс для поля `Name` таблицы `Customers`:

```
CREATE INDEX `Name` ON `Customers` (`Name`(5));
```

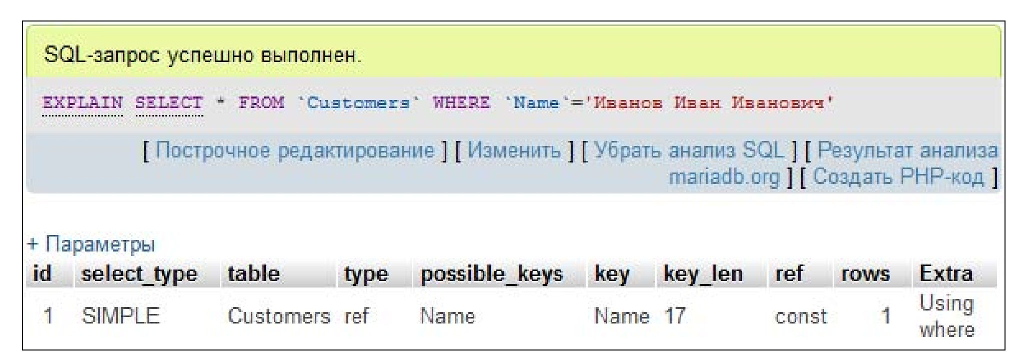
А теперь сделаем запрос и проверим его эффективность с помощью оператора `EXPLAIN`:

```
EXPLAIN SELECT * FROM `Customers` WHERE `Name`='Иванов Иван Иванович';
```

Эта команда SQL выведет результат, показанный на рис. 6.2.

Сравните результат запроса с индексом и предыдущий пример запроса без индекса.

> ***Внимание!***
>
> Значение в столбце `type` уже не равно `ALL`, а число просмотренных записей равно `1`. Это означает, что индекс полностью задействован.



SQL-запрос успешно выполнен.

EXPLAIN SELECT * FROM `Customers` WHERE `Name`='Иванов Иван Иванович'

[Построчное редактирование] [Изменить] [Убрать анализ SQL] [Результат анализа mariadb.org] [Создать PHP-код]

+ Параметры

id	select_type	table	type	possible_keys	key	key_len	ref	rows	Extra
1	SIMPLE	Customers	ref	Name	Name	17	const	1	Using where

Рис. 6.2. Результат, возвращенный командой `EXPLAIN`, после создания индекса

Индекс `FULLTEXT` применяется для полнотекстового поиска. Реализацию полнотекстового поиска и способы создания индекса `FULLTEXT` мы подробно рассмотрим в *разд. 6.10*.

Получить полную информацию об индексах таблицы позволяет SQL-команда:

```
SHOW INDEX FROM <Таблица> [FROM <База данных>];
```

Например, посмотрим, какие индексы присутствуют в таблице `Customers` нашей базы данных, для чего наберем команду:

```
SHOW INDEX FROM `Customers`;
```

Результат показан на рис. 6.3.

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation
customers	0	PRIMARY	1	id_Customer	A
customers	1	Name	1	Name	A

Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment
3	NULL	NULL		BTREE		
NULL	5	NULL		BTREE		

Рис. 6.3. Результат, возвращенный командой `SHOW INDEX`

Рассмотрим столбцы получившейся таблицы и их назначение:

- `Table` — название таблицы;
- `Non_unique` — `0`, если индекс может содержать лишь уникальные значения, `1` в противном случае;
- `Key_name` — название индекса. Для первичного — `PRIMARY`;
- `Seq_in_index` — номер последовательности столбцов в индексе, начиная с единицы;
- `Column_name` — название поля;
- `Collation` — порядок сортировки значений: `A` — по возрастанию, `NULL` — без сортировки;

- `Cardinality` — число элементов в индексе;
- `Sub_part` — для строкового поля — число символов, включаемых в индекс, для полей прочих типов — всегда `NULL`;
- `Packed` — способ упаковки индекса или `NULL`, если индекс не упакован;
- `Null` — `YES`, если индекс может включать значения `NULL`, и пустая строка в противном случае;
- `Index_type` — тип индекса. Например, `BTREE` для обычного, `FULLTEXT` — для полнотекстового;
- `Comment` — дополнительные сведения об индексе, выдаваемые самим сервером;
- `Index_comment` — примечания к индексу, заданные при его создании в SQL-запросе.

Обратите внимание: в качестве значения столбца `Cardinality` для индекса `Name` мы получили значение `NULL`. Может показаться, что в индексе нет элементов. Чтобы получить число элементов, необходимо перед использованием оператора `SHOW INDEX` выполнить SQL-команду:

```
ANALYZE TABLE <Таблица>;
```

6.4.11. Удаление таблицы и базы данных

Удалить таблицу позволяет SQL-команда:

```
DROP TABLE [IF EXISTS] <Имя таблицы>;
```

При попытке удалить несуществующую таблицу сервер выдаст сообщение об ошибке. Чтобы исключить такую ситуацию, следует вставить в указанную команду слова: `IF EXISTS`.

Удалить всю базу данных можно с помощью SQL-команды:

```
DROP DATABASE [IF EXISTS] <Имя базы данных>;
```

6.5. Доступ к базе данных MySQL из PHP-скрипта

Итак, изучение основ языка SQL закончено. Теперь мы рассмотрим функции и методы из библиотеки php_mysqli.dll, которые позволяют получить доступ к базе данных из PHP-скрипта. Чтобы можно было подключиться к MySQL из скрипта, необходимо, чтобы в файле php.ini не было символа комментария (`;`) перед строкой:

```
extension=php_mysqli.dll
```

В PHP 7.2 строка имеет следующий формат:

```
extension=mysqli
```

Библиотека php_mysqli.dll позволяет использовать как процедурный стиль доступа, так и объектный. В этом разделе мы рассмотрим оба стиля.

6.5.1. Установка соединения

Установить соединение можно двумя способами:

```
$db = mysqli_connect([<Имя хоста>[, <Имя пользователя>[, <Пароль>[,
                      <База данных>[, <Порт>[, <Сокет>]]]]]]);
$db = new mysqli([<Имя хоста>[, <Имя пользователя>[, <Пароль>[,
                  <База данных>[, <Порт>[, <Сокет>]]]]]]);
```

Все параметры здесь необязательные. Если параметры не указаны, то значения берутся из следующих директив в файле php.ini:

```
mysqli.default_host=
mysqli.default_user=
mysqli.default_pw=
mysqli.default_port=3306
mysqli.default_socket=
```

Если перед именем хоста указать комбинацию символов `p:`, то будет открыто постоянное соединение с сервером MySQL:

```
$host = 'p:localhost';
```

Закрыть соединение при *процедурном* стиле позволяет функция `mysqli_close()`:

```
mysqli_close(<Идентификатор>);
```

При *объектном* стиле используется метод `close()`:

```
<Экземпляр класса>->close();
```

Процедурный стиль возвращает идентификатор соединения, а в случае неудачи — `false`. Проверить соединение при *процедурном* стиле можно следующим образом:

```
if (@$db = mysqli_connect('localhost', 'root', '', 'tests')) {
   echo 'Подключение успешно установлено';
   // Выполняем работу с базой данных
   mysqli_close($db); // Закрываем соединение
}
else {
   echo 'Не удалось установить подключение к базе данных:<br>';
   echo 'ошибка (' . mysqli_connect_errno() . ') ';
   echo mysqli_connect_error();
}
```

В этом примере мы воспользовались следующими функциями:

- `mysqli_connect_errno()` — возвращает код ошибки или значение `0` при отсутствии ошибки;
- `mysqli_connect_error()` — возвращает строку с описанием ошибки или пустую строку при отсутствии ошибки.

Проверить соединение при *объектном* стиле можно следующим образом:

```
@$db = new mysqli('localhost', 'root', '', 'tests');
if (!mysqli_connect_errno()) {
   echo 'Подключение успешно установлено';
```

```
   // Выполняем работу с базой данных
   $db->close(); // Закрываем соединение
}
else {
   echo 'Не удалось установить подключение к базе данных:<br>';
   echo 'ошибка (' . mysqli_connect_errno() . ') ';
   echo mysqli_connect_error();
}
```

Также можно воспользоваться свойствами `connect_errno` (содержит код ошибки или значение `0`) и `connect_error` (содержит строку с описанием ошибки или пустую строку:

```
@$db = new mysqli('localhost', 'root', '', 'tests');
if (!$db->connect_errno) {
   echo 'Подключение успешно установлено';
   // Выполняем работу с базой данных
   $db->close(); // Закрываем соединение
}
else {
   echo 'Не удалось установить подключение к базе данных:<br>';
   echo 'ошибка (' . $db->connect_errno . ') ';
   echo $db->connect_error;
}
```

6.5.2. Выбор базы данных

Выбрать базу данных можно при подключении в функции `mysqli_connect()` или в конструкторе класса.

При *процедурном* стиле выбор базы данных уже после подключения осуществляет функция `mysqli_select_db()`. Формат функции:

```
mysqli_select_db(<Идентификатор>, <Имя базы данных>)
```

Функция возвращает значение `true`, если база данных успешно выбрана, и `false` — в случае неудачи:

```
if (@$db = mysqli_connect('localhost', 'root', '')) {
   if (!mysqli_select_db($db, 'tests')) {
      echo 'Не удалось выбрать базу данных';
   }
   else {
      echo 'Успешно выбрали базу данных';
      // Выполняем работу с базой данных
   }
   mysqli_close($db);
}
else {
   echo 'Не удалось установить подключение';
}
```

При *объектном* стиле используется метод `select_db()`. Формат метода:

```
<Экземпляр класса>->select_db(<Имя базы данных>)
```

Пример:

```
@$db = new mysqli('localhost', 'root', '');
if (!$db->connect_errno) {
   if (!$db->select_db('tests')) {
      echo 'Не удалось выбрать базу данных';
   }
   else {
      echo 'Успешно выбрали базу данных';
      // Выполняем работу с базой данных
   }
   $db->close();
}
else {
   echo 'Не удалось установить подключение';
}
```

6.5.3. Выполнение запроса к базе данных

Выполнить запрос к базе данных при *процедурном* стиле позволяет функция `mysqli_query()`. Формат функции:

```
mysqli_query(<Идентификатор>, <SQL-запрос>[,
             $resultmode=MYSQLI_STORE_RESULT])
```

> ***Внимание!***
> В конце SQL-запроса не следует указывать точку с запятой.

Функция возвращает идентификатор результата (для запросов типа `SELECT`), значение `true` или значение `false` в случае ошибки. Для удаления идентификатора результата и освобождения используемых ресурсов применяется функция `mysqli_free_result()`. Формат функции:

```
mysqli_free_result(<Идентификатор результата>)
```

Получить все записи таблицы `Cities` позволяет следующий код:

```
if (@$db = mysqli_connect('localhost', 'root', '', 'tests')) {
   mysqli_set_charset($db, 'utf8');
   if ($res = mysqli_query($db, 'SELECT * FROM `Cities`')) {
      // Обрабатываем извлеченные записи
      mysqli_free_result($res);
   }
   mysqli_close($db);
}
else {
   echo 'Не удалось установить подключение к базе данных';
}
```

Выполнить запрос к базе данных при *объектном* стиле позволяет метод `query()`. Формат метода:

```
<Экземпляр класса>->query(<SQL-запрос>[,
                          $resultmode=MYSQLI_STORE_RESULT])
```

Метод возвращает экземпляр результата (для запросов типа `SELECT`), значение `true` или значение `false` в случае ошибки. Для удаления экземпляра результата следует применить методы `close()`, `free()` или `free_result()`. Формат метода `free()`:

```
<Экземпляр результата>->free()
```

Получить все записи таблицы `Cities` позволяет следующий код:

```
@$db = new mysqli('localhost', 'root', '', 'tests');
if (!$db->connect_errno) {
   $db->set_charset('utf8');
   if ($res = $db->query('SELECT * FROM `Cities`')) {
      // Обрабатываем извлеченные записи
      $res->free();
   }
   $db->close();
}
else {
   echo 'Не удалось установить подключение к базе данных';
}
```

Выполнить сразу несколько запросов позволяет функция `mysqli_multi_query()` и метод `multi_query()`. В этом случае команды должны быть разделены точкой с запятой. Форматы:

```
mysqli_multi_query(<Идентификатор>, <SQL-запросы>)
<Экземпляр класса>->multi_query(<SQL-запросы>)
```

Для того чтобы записи возвращались в нужной кодировке, следует после подключения при *процедурном* стиле выполнить запрос:

```
mysqli_set_charset($db, 'cp1251');  // Для кодировки windows-1251
mysqli_set_charset($db, 'utf8');    // Для кодировки UTF-8
```

или — при *объектном* стиле:

```
$db->set_charset('cp1251');         // Для кодировки windows-1251
$db->set_charset('utf8');           // Для кодировки UTF-8
```

Можно также отправить SQL-запрос `SET NAMES`:

```
mysqli_query($db, 'SET NAMES cp1251'); // Для кодировки windows-1251
mysqli_query($db, 'SET NAMES utf8');   // Для кодировки UTF-8
```

Получить текущую кодировку можно с помощью функции `mysqli_character_set_name()` или метода `character_set_name()`:

```
echo mysqli_character_set_name($db);  // Процедурный стиль
echo $db->character_set_name();       // Объектный стиль
```

6.5.4. Обработка результата запроса при процедурном стиле

Для обработки результата запроса при процедурном стиле используются следующие функции:

- mysqli_num_rows(<Идентификатор результата>) — возвращает число записей в результате:

```
if (@$db = mysqli_connect('localhost', 'root', '', 'tests')) {
   mysqli_set_charset($db, 'utf8');
   if ($res = mysqli_query($db, 'SELECT * FROM `Cities`')) {
      echo mysqli_num_rows($res) . '<br>';
      mysqli_free_result($res);
   }
   mysqli_close($db);
}
else {
   echo 'Не удалось установить подключение к базе данных';
}
```

- mysqli_field_count(<Идентификатор соединения>) — возвращает число полей в результате последнего SQL-запроса:

```
if (@$db = mysqli_connect('localhost', 'root', '', 'tests')) {
   mysqli_set_charset($db, 'utf8');
   if ($res = mysqli_query($db, 'SELECT * FROM `Cities`')) {
      echo mysqli_field_count($db) . '<br>';
      mysqli_free_result($res);
   }
   mysqli_close($db);
}
else {
   echo 'Не удалось установить подключение к базе данных';
}
```

- mysqli_fetch_array(<Идентификатор результата>[, <Флаг>]) — возвращает результат в виде списка и (или) ассоциативного массива и сдвигает внутренний указатель на следующую запись. Если записей больше нет, возвращается значение null. Тип возвращенного результата зависит от необязательного параметра <Флаг>, который может принимать следующие значения:

  - MYSQLI_BOTH — результат в виде списка и ассоциативного массива (значение по умолчанию);
  - MYSQLI_NUM — результат в виде списка;
  - MYSQLI_ASSOC — результат в виде ассоциативного массива.

  Приведем пример, в котором сочетаются все эти варианты:

```
if (@$db = mysqli_connect('localhost', 'root', '', 'tests')) {
   mysqli_set_charset($db, 'utf8');
```

```
    if ($res = mysqli_query($db, 'SELECT * FROM `Cities`')) {
        while ($row = mysqli_fetch_array($res)) {
            echo $row[0] . ' - ' . $row['City'] . '<br>';
        }
        mysqli_free_result($res);
    }
    if ($res = mysqli_query($db, 'SELECT * FROM `Cities`')) {
        $row = mysqli_fetch_array($res, MYSQLI_NUM);
        echo $row[0] . ' - ' . $row[1] . '<br>';

        $row = mysqli_fetch_array($res, MYSQLI_ASSOC);
        echo $row['id_City'] . ' - ' . $row['City'] . '<br>';
        mysqli_free_result($res);
    }
    mysqli_close($db);
}
else {
    echo 'Не удалось установить подключение к базе данных';
}
```

- `mysqli_fetch_row(<Идентификатор результата>)` — возвращает результат в виде списка и сдвигает внутренний указатель на следующую запись. Если записей больше нет, возвращается `null`:

```
if (@$db = mysqli_connect('localhost', 'root', '', 'tests')) {
    mysqli_set_charset($db, 'utf8');
    if ($res = mysqli_query($db, 'SELECT * FROM `Cities`')) {
        while ($row = mysqli_fetch_row($res)) {
            echo $row[0] . ' - ' . $row[1] . '<br>';
        }
        mysqli_free_result($res);
    }
    mysqli_close($db);
}
else {
    echo 'Не удалось установить подключение к базе данных';
}
```

- `mysqli_fetch_assoc(<Идентификатор результата>)` — возвращает результат в виде ассоциативного массива и сдвигает внутренний указатель на следующую запись. Если записей больше нет, возвращается `null`:

```
if (@$db = mysqli_connect('localhost', 'root', '', 'tests')) {
    mysqli_set_charset($db, 'utf8');
    if ($res = mysqli_query($db, 'SELECT * FROM `Cities`')) {
        while ($row = mysqli_fetch_assoc($res)) {
            echo $row['id_City'] . ' - ' . $row['City'] . '<br>';
        }
```

```
        mysqli_free_result($res);
    }
    mysqli_close($db);
}
else {
    echo 'Не удалось установить подключение к базе данных';
}
```

- `mysqli_fetch_object(<Идентификатор результата>)` — возвращает результат в виде объекта и сдвигает внутренний указатель на следующую запись. Если записей больше нет, возвращается `null`:

```
if (@$db = mysqli_connect('localhost', 'root', '', 'tests')) {
    mysqli_set_charset($db, 'utf8');
    if ($res = mysqli_query($db, 'SELECT * FROM `Cities`')) {
        while ($row = mysqli_fetch_object($res)) {
            echo $row->id_City . ' - ' . $row->City . '<br>';
        }
        mysqli_free_result($res);
    }
    mysqli_close($db);
}
else {
    echo 'Не удалось установить подключение к базе данных';
}
```

- `mysqli_data_seek(<Идентификатор результата>, <Смещение>)` — перемещает указатель результата на выбранную строку. Нумерация строк начинается с нуля:

```
if (@$db = mysqli_connect('localhost', 'root', '', 'tests')) {
    mysqli_set_charset($db, 'utf8');
    if ($res = mysqli_query($db, 'SELECT * FROM `Cities`')) {
        mysqli_data_seek($res, 1);
        $row = mysqli_fetch_object($res);
        echo $row->id_City . ' - ' . $row->City . '<br>';
        mysqli_free_result($res);
    }
    mysqli_close($db);
}
else {
    echo 'Не удалось установить подключение к базе данных';
}
```

6.5.5. Обработка результата запроса при объектном стиле

Для обработки результата запроса при объектном стиле используются следующие методы и свойства:

- `num_rows` — содержит число записей в результате:

```
@$db = new mysqli('localhost', 'root', '', 'tests');
if (!$db->connect_errno) {
   $db->set_charset('utf8');
   if ($res = $db->query('SELECT * FROM `Cities`')) {
      echo $res->num_rows . '<br>';
      $res->free();
   }
   $db->close();
}
else {
   echo 'Не удалось установить подключение к базе данных';
}
```

- `field_count` — содержит число полей в результате:

```
@$db = new mysqli('localhost', 'root', '', 'tests');
if (!$db->connect_errno) {
   $db->set_charset('utf8');
   if ($res = $db->query('SELECT * FROM `Cities`')) {
      echo $res->field_count . '<br>';
      $res->free();
   }
   $db->close();
}
else {
   echo 'Не удалось установить подключение к базе данных';
}
```

- `fetch_array([<Флаг>])` — возвращает результат в виде списка и (или) ассоциативного массива и сдвигает внутренний указатель на следующую запись. Если записей больше нет, возвращается значение `null`. Тип возвращенного результата зависит от необязательного параметра `<Флаг>`, который может принимать следующие значения:
  - `MYSQLI_BOTH` — результат в виде списка и ассоциативного массива (значение по умолчанию);
  - `MYSQLI_NUM` — результат в виде списка;
  - `MYSQLI_ASSOC` — результат в виде ассоциативного массива.

  Следующий пример иллюстрирует все эти варианты:

```
@$db = new mysqli('localhost', 'root', '', 'tests');
if (!$db->connect_errno) {
   $db->set_charset('utf8');
   if ($res = $db->query('SELECT * FROM `Cities`')) {
      while ($row = $res->fetch_array()) {
         echo $row[0] . ' - ' . $row['City'] . '<br>';
      }
```

```
      $res->free();
   }
   if ($res = mysqli_query($db, 'SELECT * FROM `Cities`')) {
      $row = $res->fetch_array(MYSQLI_NUM);
      echo $row[0] . ' - ' . $row[1] . '<br>';

      $row = $res->fetch_array(MYSQLI_ASSOC);
      echo $row['id_City'] . ' - ' . $row['City'] . '<br>';
      $res->free();
   }
   $db->close();
}
else {
   echo 'Не удалось установить подключение к базе данных';
}
```

- `fetch_row()` — возвращает результат в виде списка и сдвигает внутренний указатель на следующую запись. Если записей больше нет, возвращается `null`:

```
@$db = new mysqli('localhost', 'root', '', 'tests');
if (!$db->connect_errno) {
   $db->set_charset('utf8');
   if ($res = $db->query('SELECT * FROM `Cities`')) {
      while ($row = $res->fetch_row()) {
         echo $row[0] . ' - ' . $row[1] . '<br>';
      }
      $res->free();
   }
   $db->close();
}
else {
   echo 'Не удалось установить подключение к базе данных';
}
```

- `fetch_assoc()` — возвращает результат в виде ассоциативного массива и сдвигает внутренний указатель на следующую запись. Если записей больше нет, возвращается `null`:

```
@$db = new mysqli('localhost', 'root', '', 'tests');
if (!$db->connect_errno) {
   $db->set_charset('utf8');
   if ($res = $db->query('SELECT * FROM `Cities`')) {
      while ($row = $res->fetch_assoc()) {
         echo $row['id_City'] . ' - ' . $row['City'] . '<br>';
      }
      $res->free();
   }
   $db->close();
}
```

```
else {
   echo 'Не удалось установить подключение к базе данных';
}
```

- `fetch_object()` — возвращает результат в виде объекта и сдвигает внутренний указатель на следующую запись. Если записей больше нет, возвращается `null`:

```
@$db = new mysqli('localhost', 'root', '', 'tests');
if (!$db->connect_errno) {
   $db->set_charset('utf8');
   if ($res = $db->query('SELECT * FROM `Cities`')) {
      while ($row = $res->fetch_object()) {
         echo $row->id_City . ' - ' . $row->City . '<br>';
      }
      $res->free();
   }
   $db->close();
}
else {
   echo 'Не удалось установить подключение к базе данных';
}
```

- `data_seek(<Смещение>)` — перемещает указатель результата на выбранную строку. Нумерация строк начинается с нуля:

```
@$db = new mysqli('localhost', 'root', '', 'tests');
if (!$db->connect_errno) {
   $db->set_charset('utf8');
   if ($res = $db->query('SELECT * FROM `Cities`')) {
      $res->data_seek(1);
      $row = $res->fetch_object();
      echo $row->id_City . ' - ' . $row->City . '<br>';
      $res->free();
   }
   $db->close();
}
else {
   echo 'Не удалось установить подключение к базе данных';
}
```

6.5.6. Экранирование специальных символов

Функция `mysqli_real_escape_string(<Идентификатор соединения>, <Строка>)` в процедурном стиле и метод `real_escape_string(<Строка>)` в объектном стиле экранируют все специальные символы в строке, учитывая кодировку соединения, и возвращают строку, которую можно безопасно использовать в SQL-запросах (листинг 6.1).

Листинг 6.1. Экранирование специальных символов

```
$new_city = "Д'Арк"; // Такие данные передаются из формы
if (@$db = mysqli_connect('localhost', 'root', '', 'tests')) {
   mysqli_set_charset($db, 'utf8');
   // Экранируем спецсимволы
   $new_city = mysqli_real_escape_string($db, $new_city);
   $query = "INSERT INTO `Cities` VALUES(NULL, '$new_city')";
   if (!mysqli_query($db, $query)) {
      echo 'Ошибка: ' . mysqli_error($db);
   }
   if ($res = mysqli_query($db, 'SELECT * FROM `Cities`')) {
      while ($row = mysqli_fetch_row($res)) {
         echo $row[0] . ' - ' . $row[1] . '<br>';
      }
      mysqli_free_result($res);
   }
   mysqli_close($db);
}
else {
   echo 'Не удалось установить подключение к базе данных';
}
```

> ***Внимание!***
>
> Никогда напрямую не передавайте в SQL-запрос данные, полученные из полей формы. Это потенциальная угроза безопасности. Всегда применяйте функцию `mysqli_real_escape_string()` или метод `real_escape_string()`.

Рассмотрим проблему, возникающую, если не применить функцию `mysqli_real_escape_string()` или метод `real_escape_string()` для входных данных.

Создадим таблицу `user` и добавим в нее две записи:

```
CREATE TABLE `user` (
  `id_user` INT(9) AUTO_INCREMENT,
  `login` VARCHAR(50),
  `passw` VARCHAR(32),
  PRIMARY KEY (`id_user`)
) ENGINE=MyISAM CHARSET=utf8 COLLATE utf8_general_ci;
INSERT INTO `user` VALUES (NULL, 'Admin', '123');
INSERT INTO `user` VALUES (NULL, 'Nik', '456');
```

Теперь инсценируем вход злоумышленника в систему под логином администратора (листинг 6.2). При этом злоумышленник даже не должен знать его пароль.

Листинг 6.2. Вход злоумышленника в систему под логином администратора

```
<?php
// Никогда так не делайте!!!
// Такие данные пришли из формы:
$_POST['login'] = "' OR ''='";
```

```
$_POST['passw'] = "' OR ''='";
$login = $_POST['login'];
$passw = $_POST['passw'];
if (@$db = mysqli_connect('localhost', 'root', '', 'tests')) {
   mysqli_set_charset($db, 'utf8');
   $query = "SELECT * FROM `user` WHERE `login`='$login' ";
   $query .= "AND `passw`='$passw'";
   echo htmlspecialchars($query) . '<br>';
   if ($res = mysqli_query($db, $query)) {
      if (mysqli_num_rows($res) > 0) {
         echo 'Полный доступ в систему!!!<br>';
      }
      while ($row = mysqli_fetch_row($res)) {
         echo $row[1] . '<br>';
      }
      mysqli_free_result($res);
   }
   mysqli_close($db);
}
else {
   echo 'Не удалось установить подключение к базе данных';
}
```

Введя указанные в начале примера строки в форме, злоумышленник получит:

```
SELECT * FROM `user` WHERE `login`='' OR ''='' AND `passw`='' OR ''=''
Полный доступ в систему!!!
Admin
Nik
```

Как можно видеть, злоумышленник вошел в систему, не зная пароля. В этом примере, т. к. учетная запись администратора расположена на первой позиции, он вошел под записью администратора — ему просто повезло. А теперь покажем, как он войдет в систему именно под учетной записью администратора. Для этого входящие данные изменим на:

```
$_POST['login'] = "Admin'/*";
$_POST['passw'] = "*/ '";
```

После выполнения скрипта получим следующий результат:

```
SELECT * FROM `user` WHERE `login`='Admin'/*' AND `passw`='*/ ''
Полный доступ в систему!!!
Admin
```

Как видно из результата, он является администратором. Все, что расположено между `/*` и `*/` — это комментарий. В итоге SQL-запрос будет выглядеть так:

```
SELECT * FROM `user` WHERE `login`='Admin' ''
```

Пароль в данном случае вообще не проверяется. Таким образом, достаточно знать логин пользователя и можно войти без пароля!

При обработке данных функцией `mysqli_real_escape_string()` или методом `real_escape_string()` такого бы не случилось:

```
$login = mysqli_real_escape_string($db, $login);
$passw = mysqli_real_escape_string($db, $passw);
$query = "SELECT * FROM `user` WHERE `login`='$login' ";
$query .= "AND `passw`='$passw'";
echo htmlspecialchars($query) . '<br>';
```

В первом случае скрипт выведет только:

```
SELECT * FROM `user` WHERE `login`='\' OR \'\'=\''
AND `passw`='\' OR \'\'=\''
```

А во втором:

```
SELECT * FROM `user` WHERE `login`='Admin\'/*' AND `passw`='*/ \''
```

В результате все опасные символы были экранированы.

> ***Примечание***
>
> Выводить код SQL напрямую в Web-страницу также не рекомендуется, т. к. это дает злоумышленнику информацию о структуре базы данных.

6.6. Транзакции

При работе с базами данных очень часто бывает необходимо выполнять какие-либо сложные действия, включающие несколько операций по добавлению, изменению и удалению записей. И эти операции обязательно должны быть либо выполнены все и полностью, либо, если в процессе их обработки произойдет сбой, не выполнены все и полностью, — в противном случае нарушится целостность данных, хранящихся в базе. Например, при выписывании расходной ведомости товар списывается со склада. Если во время списания произойдет ошибка, то расчетная ведомость будет сформирована, а товар со склада списан не будет. Чтобы гарантировать успешное выполнение группы запросов, используется механизм *транзакций*.

> ***Внимание!***
>
> Следует учитывать, что транзакции поддерживаются только таблицами, имеющими тип `InnoDB`. Таблицы типа `MyISAM` транзакции не поддерживают.

Давайте создадим в базе данных `tests` две таблицы: первая (`inv_goods`) — будет хранить список товаров и их количество, имеющееся на складе, а вторая (`inv_cns`) — список расходных ведомостей, документирующих передачу товаров со склада заказчикам:

```
CREATE TABLE `inv_goods` (
   `id_good` INT NOT NULL AUTO_INCREMENT,
   `name` VARCHAR(10),
   `count` SMALLINT,
   PRIMARY KEY (`id_good`)
) ENGINE=InnoDB CHARSET=utf8 COLLATE utf8_general_ci;
```

```
CREATE TABLE `inv_cns` (
   `id_cn` INT NOT NULL AUTO_INCREMENT,
   `id_good` INT,
   `count` SMALLINT,
   PRIMARY KEY (`id_cn`),
   FOREIGN KEY (`id_good`) REFERENCES `inv_goods` (`id_good`)
   ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB CHARSET=utf8 COLLATE utf8_general_ci;
```

Добавим пару записей в таблицу `inv_goods`:

```
INSERT INTO `inv_goods` VALUES (NULL, 'Мыло', 20);
INSERT INTO `inv_goods` VALUES (NULL, 'Шило', 10);
```

6.6.1. Автозавершение транзакций и его отключение

По умолчанию каждая операция запускается в составе транзакции, даже если мы не указали это явно. Такие создаваемые по умолчанию транзакции подтверждаются автоматически (*автозавершение* транзакций):

```
/* Запускается транзакция по умолчанию */
SELECT * from `inv_goods`;
/* Транзакция по умолчанию подтверждается */
/* Запускается транзакция по умолчанию */
INSERT INTO `inv_goods` VALUES (NULL, 'Молоток', 5);
/* Транзакция по умолчанию подтверждается */
```

Как видим, это относится, в том числе, и к операциям выборки данных.

Управлять автозавершением транзакций позволяют следующие SQL-команды:

- `SET autocommit=0;` — отключает автозавершение транзакций. В этом случае мы должны явным образом подтвердить (`COMMIT`) или откатить изменения (`ROLLBACK`):

  ```
  SET autocommit=0;
  /* Автозавершение транзакций отключено */
  INSERT INTO `inv_goods` VALUES (NULL, 'Швабра', 12);
  /* Выполняем завершение транзакции */
  COMMIT;
  ```

- `SET autocommit=1;` — включает автозавершение транзакций;
- `SELECT @@autocommit;` — позволяет узнать текущий статус автозавершения транзакций.

6.6.2. Запуск, подтверждение и отмена транзакций

Управлять транзакциями позволяют следующие SQL-команды:

- `START TRANSACTION` — запускает транзакцию. Все операции выборки, добавления, изменения и удаления записей, следующие после этой команды, будут выполняться в составе транзакции. После явного запуска транзакции автозавершение временно отключается до вызова `COMMIT` или `ROLLBACK`.

Вот пример запуска транзакции:

```
START TRANSACTION;
```

После ключевых слов START TRANSACTION могут быть указаны следующие конструкции:

- WITH CONSISTENT SNAPSHOT — создать «снимок» данных непосредственно при запуске транзакции;
- READ WRITE — чтение и запись;
- READ ONLY — только чтение;

- COMMIT — подтверждает транзакцию и завершает ее. При получении этой команды сервер выполняет все команды изменения записей, что находились между ней и предыдущей командой запуска транзакции, и подтверждает все сделанные этими командами изменения.

  Вот пример подтверждения транзакции:

  ```
  COMMIT;
  ```

- ROLLBACK — отменяет все изменения в рамках транзакции и завершает транзакцию (выполняет откат транзакции).

  Вот пример отката транзакции:

  ```
  ROLLBACK;
  ```

  После ключевых слов COMMIT и ROLLBACK могут быть указаны следующие конструкции:

  ```
  COMMIT [AND [NO] CHAIN] [[NO] RELEASE];
  ROLLBACK [AND [NO] CHAIN] [[NO] RELEASE];
  ```

- AND CHAIN — новая транзакция начинается, как только текущая заканчивается. Новая транзакция будет иметь точно такой же уровень изоляции и режим доступа (READ WRITE или READ ONLY);
- AND NO CHAIN — отключает CHAIN;
- RELEASE — текущий сеанс клиента отключается после завершения транзакции;
- NO RELEASE — отключает RELEASE.

Узнать режим по умолчанию позволяет системная переменная completion_type:

```
SELECT @@completion_type;
/* Выведет: NO_CHAIN */
```

Выясним, какой идентификатор получил товар «Мыло»:

```
SELECT `id_good` FROM `inv_goods` WHERE `name` = 'Мыло';
/* Выведет: 1 */
```

Теперь давайте попробуем выписать расходную ведомость на отпуск трех единиц этого товара. Для этого нам придется:

- добавить в таблицу inv_cns запись, представляющую саму расходную ведомость;

❒ уменьшить значение поля `count` таблицы `inv_goods` на три, чтобы показать уменьшение хранящегося на складе товара.

Поскольку обе эти операции обязательно должны быть выполнены все и полностью (в противном случае, если случится сбой, мы получим недостачу товара на складе), мы заключим их в транзакцию и не забудем подтвердить ее:

```
START TRANSACTION;
INSERT INTO `inv_cns` VALUES (NULL, 1, 3);
UPDATE `inv_goods` SET `count`=`count`-3 WHERE `id_good`=1;
COMMIT;
```

Посмотрим содержимое таблицы `inv_cns`:

```
SELECT * FROM `inv_cns`;
```

Мы увидим, что в таблице появилась одна запись — представляющая только что выписанную нами расходную ведомость.

Проверим, произошла ли выдача товара со склада:

```
SELECT `count` FROM `inv_goods` WHERE `name`='Мыло';
/* Выведет: 17 */
```

Как видим, товар был успешно выдан.

Предположим, что мы собирались выписать расходную ведомость на две единицы товара «Шило», но в последний момент передумали и решили отменить выдачу, выполнив откат транзакции:

```
START TRANSACTION;
SELECT @inv_id:=`id_good` FROM `inv_goods` WHERE `name`='Шило';
INSERT INTO `inv_cns` VALUES (NULL, @inv_id, 2);
UPDATE `inv_goods` SET `count`=`count`-2 WHERE `id_good`=@inv_id;
ROLLBACK;
```

В этом случае все операции, заключенные в транзакцию, не будут выполнены. В чем мы убедимся, просмотрев содержимое таблиц `inv_goods` и `inv_cns`.

Отметим, что в последнем примере для временного хранения идентификатора товара, выдаваемого со склада, мы использовали переменную SQL.

Если в процессе обработки включенных в состав транзакции команд возникнет ошибка, то эту ошибку вы должны обработать в программе и явным образом откатить транзакцию с помощью команды `ROLLBACK`. Не стоит надеяться на то, что команды отменятся автоматически. Если команды выполнить в программе phpMyAdmin, то при ошибке откат будет выполнен, но если не предусмотреть обработку ошибок в своей программе, то никакого отката сделано не будет! Например, закомментируйте все инструкции `throw` в листинге 6.3 и добавьте ошибку в название поля (например, `id_good2`) в команде `UPDATE`. В результате расходная ведомость будет сформирована, а вот списания товара со склада не произойдет!

Следует также учитывать, что внутри транзакции можно без проблем работать с командами `SELECT`, `INSERT`, `UPDATE` и `DELETE`, но другие команды либо невозможно

откатить, либо они подтверждают транзакцию автоматически. Например, команду DROP TABLE нельзя откатить, а команда CREATE TABLE — неявно завершает транзакцию, как если бы была вызвана команда COMMIT. Поэтому такие команды лучше вынести за рамки транзакции, иначе выполнить откат других команд, возможно, не получится.

6.6.3. Изоляция транзакций

Поскольку к базе данных MySQL одновременно могут подключаться сразу несколько пользователей, в ней могут выполняться сразу несколько транзакций, запущенных разными пользователями. И эти транзакции могут изменять содержимое одной и той же таблицы.

Введение в изоляцию транзакций

В связи с этим возникает вопрос: как сделанные в таблице изменения отслеживаются командами SELECT, находящимися в составе транзакции? В этом случае MySQL соблюдает следующие правила:

- при выполнении первой команды SELECT, присутствующей в транзакции, в памяти компьютера создается своего рода «снимок» данных, являющихся актуальными на момент выполнения этой операции. Все последующие команды SELECT, что включены в транзакцию, оперируют именно этим «снимком»;
- «снимок» отслеживает все действия команд INSERT, UPDATE и DELETE, находящихся в той же транзакции, где он был создан;
- действия команд INSERT, UPDATE и DELETE, присутствующих в других транзакциях, «снимок» не отслеживает.

Как видим, каждая транзакция работает со своей копией данных. Такой подход называется *изоляцией транзакций*.

Если в транзакции присутствует команда выборки данных SELECT, ее рекомендуется поставить самой первой, непосредственно после команды START TRANSACTION. В этом случае «снимок» данных будет создан сразу же после запуска транзакции.

Как вариант, можно использовать команду START TRANSACTION WITH CONSISTENT SNAPSHOT — она тоже указывает MySQL создать «снимок» данных непосредственно при запуске транзакции:

```
START TRANSACTION WITH CONSISTENT SNAPSHOT;
/* Команды транзакции */
COMMIT | ROLLBACK;
```

Уровни изоляции транзакций

MySQL поддерживает четыре уровня изоляции транзакции, характеризующих степень доступности таблиц, вовлеченных в транзакцию, для других транзакций, которые запущены параллельно с текущей. Для указания уровня изоляции применяется команда SET TRANSACTION ISOLATION LEVEL:

```
SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL
REPEATABLE READ | READ COMMITTED | READ UNCOMMITTED |
SERIALIZABLE
```

Рассмотрим все доступные нам уровни изоляции транзакций (в порядке увеличения степени надежности):

- `READ UNCOMMITTED` — транзакции не изолированы друг от друга, и команда `SELECT` может прочитать данные, в которые были внесены изменения параллельно работающими транзакциями. Вследствие этого может возникнуть ситуация, когда одна транзакция может прочитать данные, измененные другой транзакцией, после чего — в случае отката последней — первая транзакция будет оперировать данными, которых фактически нет в таблице. Самый низкий уровень изоляции;
- `READ COMMITTED` — каждая команда `SELECT` в транзакции создает свой собственный «снимок» данных, которым и оперирует. Транзакции изолированы друг от друга;
- `REPEATABLE READ` — первая команда `SELECT` в транзакции создает «снимок» данных, который используется всеми последующими командами такого же типа. Транзакции изолированы друг от друга. Это уровень изоляции по умолчанию;
- `SERIALIZABLE` — похож на `REPEATABLE READ`, но, если автозавершение транзакций отключено, каждая команда `SELECT` временно блокирует таблицы, из которых выбирает данные, с тем, чтобы никакая другая транзакция не смогла получить к ним доступ. Самый надежный уровень изоляции, но при этом самый медленный.

В большинстве случаев уровень изоляции `REPEATABLE READ`, используемый по умолчанию, является лучшим выбором:

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
START TRANSACTION;
/* Команды транзакции */
COMMIT;
```

Кроме того, мы можем указать, на какие транзакции будет распространяться действие команды `SET TRANSACTION ISOLATION LEVEL`:

- если не указаны ключевые слова `GLOBAL` и `SESSION`, действие команды распространится лишь на следующую транзакцию. Все транзакции, запущенные после нее, получат уровень изоляции по умолчанию:

  ```
  SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
  START TRANSACTION;
  /* Эта транзакция использует уровень изоляции READ COMMITTED */
  COMMIT;
  START TRANSACTION;
  /* Эта транзакция будет запущена с уровнем изоляции
  по умолчанию - REPEATABLE READ */
  COMMIT;
  ```

- если указано ключевое слово `SESSION`, действие команды распространится на все транзакции, запущенные в течение текущей сессии;
- если указано ключевое слово `GLOBAL`, действие команды распространится на все транзакции, запущенные в течение текущей и последующих сессий.

6.6.4. Именованные точки сохранения

Внутри транзакции можно создать именованную метку, называемую *точкой сохранения*. Для этого предназначена команда `SAVEPOINT`:

```
SAVEPOINT <Метка>;
```

Если транзакция уже имеет точку сохранения с указанным именем, то старая точка удаляется и устанавливается новая.

Для удаления точки сохранения предназначена команда `RELEASE`:

```
RELEASE SAVEPOINT <Метка>;
```

При этом не происходит подтверждение или откат. Все точки сохранения удаляются при завершении транзакции с помощью `COMMIT` или `ROLLBACK` без указания метки.

Чтобы отменить изменения, выполненные после метки, используется команда `ROLLBACK`:

```
ROLLBACK TO SAVEPOINT <Метка>;
```

Пример:

```
START TRANSACTION;
SAVEPOINT MyLabel_1;
INSERT INTO `inv_goods` VALUES (NULL, 'Вешалка', 15);
RELEASE SAVEPOINT MyLabel_1;
SAVEPOINT MyLabel_2;
INSERT INTO `inv_goods` VALUES (NULL, 'Порошок', 20);
ROLLBACK TO SAVEPOINT MyLabel_2;
/* Будет добавлена только вешалка */
COMMIT;
```

6.6.5. Блокировка таблиц и строк

Итак, мы научились выполнять группу команд как единое целое в составе транзакции. Однако, пока мы запрашиваем количество товара на складе, параллельно аналогичное действие может совершать и другой процесс. Представьте: два процесса видят пять единиц товара и одновременно оформляют расходные ведомости на это количество и списывают товар со склада. В итоге количество товара на складе станет отрицательным. Как успеть списать товар раньше, чем его успеет списать параллельный процесс, и при этом не дать этому процессу списать то, чего уже нет? Для этого нужно воспользоваться *блокировками*. Таблицы типа `InnoDB` поддерживают блокировки на уровне таблиц и строк.

Блокировка отдельных таблиц осуществляется с помощью команды `LOCK TABLES`, а снять блокировку позволяет команда `UNLOCK TABLES`. Блокировку можно установить только на чтение (`READ`) или на чтение и запись (`WRITE`). При работе с транзакциями следует учитывать, что эти команды автоматически завершают текущую транзакцию. Поэтому вместо команды `START TRANSACTION` следует отключить автоматическое завершение транзакций:

```
SET autocommit=0;
LOCK TABLES `inv_goods` WRITE, `inv_cns` WRITE;
SELECT @inv_id:=`id_good` FROM `inv_goods` WHERE `name`='Мыло';
INSERT INTO `inv_cns` VALUES (NULL, @inv_id, 2);
UPDATE `inv_goods` SET `count`=`count`-2 WHERE `id_good`=@inv_id;
COMMIT;
UNLOCK TABLES;
SET autocommit=1;
```

Блокировка таблиц может заметно сказаться на производительности — ведь другие процессы будут ждать снятия блокировки. В нашем случае более правильно будет использовать блокировку на уровне строк. Для этого после команды `SELECT` нужно добавить конструкцию `FOR UPDATE`. В этом случае будут заблокированы только строки, которые возвращает эта команда. Блокировка снимается при завершении транзакции:

```
START TRANSACTION;
SELECT @inv_id:=`id_good` FROM `inv_goods`
WHERE `name`='Мыло' LIMIT 1 FOR UPDATE;
INSERT INTO `inv_cns` VALUES (NULL, @inv_id, 2);
UPDATE `inv_goods` SET `count`=`count`-2 WHERE `id_good`=@inv_id;
COMMIT;
```

> ***Примечание***
>
> В этих примерах мы не проверяем количество товара на складе только для упрощения их демонстрации. На практике это нужно делать обязательно. Кроме того, редко используется выборка по наименованию товара, практически всегда указывается уникальный идентификатор товара в таблице.

Иногда нужно не изменять значение в поле, а просто иметь гарантию, что при добавлении записи в другую таблицу связанная запись из первой таблицы не будет удалена параллельным процессом. Для этого после команды `SELECT` нужно добавить конструкцию `LOCK IN SHARE MODE`. В этом случае будут заблокированы для изменения только строки, которые возвращает эта команда. Блокировка снимается при завершении транзакции.

Давайте добавим в базу данных `tests` еще одну таблицу с какой-нибудь дополнительной информацией о товаре из таблицы `inv_goods`:

```
CREATE TABLE `inv_info` (
   `id_info` INT NOT NULL AUTO_INCREMENT,
   `id_good` INT,
   `text_info` TEXT,
```

```
   PRIMARY KEY (`id_info`),
   FOREIGN KEY (`id_good`) REFERENCES `inv_goods` (`id_good`)
   ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB CHARSET=utf8 COLLATE utf8_general_ci;
```

Теперь добавим новую запись в таблицу, при этом обеспечивая ссылочную целостность с существующим товаром:

```
START TRANSACTION;
SELECT @inv_id:=`id_good` FROM `inv_goods`
WHERE `name`='Мыло' LIMIT 1 LOCK IN SHARE MODE;
INSERT INTO `inv_info` VALUES (NULL, @inv_id, 'Отличное мыло');
COMMIT;
```

6.6.6. Поддержка транзакций библиотекой *php_mysqli.dll*

Для управления транзакциями при *процедурном* стиле в библиотеке php_mysqli.dll предназначены следующие функции:

- `mysqli_begin_transaction(<Идентификатор>[, <Флаг>[, <Метка>]])` — запускает транзакцию. В параметре `<Флаг>` можно указать константы `MYSQLI_TRANS_START_READ_WRITE`, `MYSQLI_TRANS_START_READ_ONLY` или `MYSQLI_TRANS_START_WITH_CONSISTENT_SNAPSHOT`;
- `mysqli_commit(<Идентификатор>[, <Флаги>[, <Метка>]])` — подтверждает транзакцию;
- `mysqli_rollback(<Идентификатор>[, <Флаги>[, <Метка>]])` — отменяет транзакцию.

  В параметре `<Флаги>` в функциях `mysqli_commit()` и `mysqli_rollback()` можно указать константы `MYSQLI_TRANS_COR_RELEASE`, `MYSQLI_TRANS_COR_NO_RELEASE`, `MYSQLI_TRANS_COR_AND_CHAIN` и `MYSQLI_TRANS_COR_AND_NO_CHAIN`;
- `mysqli_autocommit(<Идентификатор>, <true | false>)` — включает (`true`) или отключает (`false`) автозавершение транзакций.

При *объектном* стиле используются следующие методы:

- `begin_transaction([<Флаг>[, <Метка>]])` — запускает транзакцию. В параметре `<Флаг>` можно указать константы `MYSQLI_TRANS_START_READ_WRITE`, `MYSQLI_TRANS_START_READ_ONLY` или `MYSQLI_TRANS_START_WITH_CONSISTENT_SNAPSHOT`;
- `commit([<Флаги>[, <Метка>]])` — подтверждает транзакцию;
- `rollback([<Флаги>[, <Метка>]])` — отменяет транзакцию.

  В параметре `<Флаги>` в методах `commit()` и `rollback()` можно указать константы `MYSQLI_TRANS_COR_RELEASE`, `MYSQLI_TRANS_COR_NO_RELEASE`, `MYSQLI_TRANS_COR_AND_CHAIN` и `MYSQLI_TRANS_COR_AND_NO_CHAIN`;
- `autocommit(<true | false>)` — включает (`true`) или отключает (`false`) автозавершение транзакций.

Пример обработки ошибок в рамках транзакции, приведен в листинге 6.3.

Листинг 6.3. Управление транзакцией при процедурном стиле

```
$c = 2;       // Требуемое количество
$inv_id = 1;  // ID товара
if (@$db = mysqli_connect('localhost', 'root', '', 'tests')) {
   mysqli_set_charset($db, 'utf8');
   mysqli_query($db,
                'SET TRANSACTION ISOLATION LEVEL REPEATABLE READ');
   mysqli_begin_transaction($db,
                MYSQLI_TRANS_START_WITH_CONSISTENT_SNAPSHOT);
   $q1 =  "SELECT `count` FROM `inv_goods` ";
   $q1 .= "WHERE `id_good`={$inv_id} FOR UPDATE";
   $q2 =  "INSERT INTO `inv_cns` VALUES (NULL, $inv_id, $c)";
   $q3 =  "UPDATE `inv_goods` SET `count`=`count`-{$c} ";
   $q3 .= "WHERE `id_good`={$inv_id}";
   try {
      if ($res = mysqli_query($db, $q1)) {
         if (mysqli_num_rows($res) != 1) {
            mysqli_free_result($res);
            throw new \Exception('Не одна запись');
         }
         $row = mysqli_fetch_assoc($res);
         $count = intval($row['count'] ?? 0);
         if ($count < $c) {
            mysqli_free_result($res);
            throw new \Exception('Недостаточно товара');
         }
         mysqli_free_result($res);
      }
      else throw new \Exception(mysqli_error($db));
      if (!mysqli_query($db, $q2)) {
         throw new \Exception(mysqli_error($db));
      }
      if (!mysqli_query($db, $q3)) {
         throw new \Exception(mysqli_error($db));
      }
      if (mysqli_commit($db)) {
         echo 'Транзакция успешно подтверждена';
      }
      else throw new \Exception(mysqli_error($db));
   } catch (\Exception $e) {
      echo 'Ошибка: ' . $e->getMessage();
      mysqli_rollback($db);
   }
```

```
   mysqli_close($db);
}
else {
   echo 'Не удалось установить подключение к базе данных';
}
```

6.7. Операторы MySQL

Операторы позволяют выполнить с данными определенные действия. Например, математические операторы предназначены для арифметических вычислений. Рассмотрим операторы, доступные в MySQL.

Выполнять SQL-команды мы будем в программе MySQL Command Line Client, а точнее — в ее аналоге MariaDB Command Line Client. Программа является консольной, поэтому запускаем приложение **Командная строка** и переходим в каталог C:\xampp\mysql\bin:

```
C:\Users\Unicross>cd C:\xampp\mysql\bin

C:\xampp\mysql\bin>
```

Консоль по умолчанию работает с кодировкой windows-866. Чтобы убедиться в этом, набираем команду:

```
C:\xampp\mysql\bin>chcp
Текущая кодовая страница: 866
```

Мы собираемся работать с кодировкой windows-1251, поэтому набираем такую команду:

```
C:\xampp\mysql\bin>chcp 1251
Текущая кодовая страница: 1251
```

Если в последней строке русские буквы исказились, то нужно сменить шрифт. Для этого щелкаем правой кнопкой мыши на заголовке окна и из контекстного меню выбираем пункт **Свойства**. В открывшемся окне переходим на вкладку **Шрифт** и в списке **Шрифт** выбираем пункт **Lucida Console**. В списке **Размер** выбираем пункт **12**. Нажимаем кнопку **OK**.

Запускаем программу с помощью команды:

```
mysql -u root -p
```

Программа выведет запрос на ввод пароля. Мы не задавали пароль для пользователя `root`, поэтому просто нажимаем клавишу <Enter>. В случае успешного входа отобразится приветствие сервера, и программа перейдет в режим ожидания команд:

```
C:\xampp\mysql\bin>mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 12
Server version: 10.1.29-MariaDB mariadb.org binary distribution
```

```
Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input
statement.

MariaDB [(none)]>
```

Слово `none` внутри скобок говорит о том, что база данных не выбрана. Выбираем базу данных `tests`:

```
MariaDB [(none)]> USE `tests`;
Database changed
MariaDB [tests]>
```

Давайте сразу зададим кодировку соединения:

```
MariaDB [tests]> SET NAMES cp1251;
Query OK, 0 rows affected (0.00 sec)
```

Каждая команда должна завершаться точкой с запятой. Если не указать точку с запятой и нажать клавишу <Enter>, то программа выведет приглашение для продолжения ввода команды `->`.

В качестве примера выведем содержимое таблицы `Cities`:

```
MariaDB [tests]> SELECT *
    -> FROM `Cities`;
+---------+-----------------+
| id_City | City            |
+---------+-----------------+
|       1 | Санкт-Петербург |
|       2 | Москва          |
|       3 | Д'Арк           |
+---------+-----------------+
3 rows in set (0.00 sec)
```

Если вместо точки с запятой указать комбинацию `\G`, то результат будет выведен не в виде таблицы, а в виде списка:

```
MariaDB [tests]> SELECT * FROM `Cities`\G
*************************** 1. row ***************************
id_City: 1
   City: Санкт-Петербург
*************************** 2. row ***************************
id_City: 2
   City: Москва
*************************** 3. row ***************************
id_City: 3
   City: Д'Арк
3 rows in set (0.00 sec)
```

Для завершения работы программы нужно ввести команду:

```
QUIT;
```

6.7.1. Математические операторы

Приведем список математических операторов:

- `+` — сложение:

  ```
  SELECT 8 + 5;
  ```

- `-` — вычитание:

  ```
  SELECT 10 - 5;
  ```

- `*` — умножение:

  ```
  SELECT 10 * 5;
  ```

- `/` — деление:

  ```
  SELECT 10 / 5;
  /* Выведет: 2.0000 */
  ```

- `DIV` — целочисленное деление:

  ```
  SELECT 10 DIV 5;
  /* Выведет: 2 */
  SELECT 10 DIV 3;
  /* Выведет: 3 */
  ```

- `%` и `MOD` — остаток от деления:

  ```
  SELECT 10 % 2;
  /* Выведет: 0 */
  SELECT 9 % 2;
  /* Выведет: 1 */
  SELECT 10 MOD 2;
  /* Выведет: 0 */
  ```

  Вместо операторов `%` и `MOD` можно использовать функцию `MOD()`:

  ```
  SELECT MOD(10, 2);
  /* Выведет: 0 */
  ```

Следует отметить, что если один из операндов равен `NULL`, то результат операции также будет равен `NULL`. В отличие от языков программирования, деление на ноль здесь не приводит к генерации сообщения об ошибке, — результатом операции деления на ноль является значение `NULL`:

```
SELECT 10 * NULL;
/* Выведет: NULL */
SELECT 10 / 0;
/* Выведет: NULL */
```

Если необходимо сменить знак числа, то перед операндом следует указать символ `-` (минус):

```
SELECT -(-5);
/* Выведет: 5 */
```

В качестве примера рассмотрим возможность подсчета переходов по рекламной ссылке. Для этого создадим таблицу `counter` в базе данных `tests`:

```
CREATE TABLE `counter` (
   `id_link` INT NOT NULL AUTO_INCREMENT,
   `total` INT,
   PRIMARY KEY (`id_link`)
) ENGINE=MyISAM CHARSET=utf8 COLLATE utf8_general_ci;
```

Затем добавим одну запись:

```
INSERT INTO `counter` VALUES (1, 0);
```

Для подсчета переходов в тексте ссылки укажем идентификатор в базе данных и URL-адрес:

```
<a href="go.php?id=1&amp;url=http://www.mail.ru/">Перейти</a>
```

Переходы регистрируются в файле go.php. Исходный код файла приведен в листинге 6.4.

Листинг 6.4. Регистрация переходов по ссылке

```
<?php
if (!isset($_GET['id']) || !isset($_GET['url'])) die('Ошибка');
$id = intval($_GET['id']);
if ($id > 0) {
   if (@$db = mysqli_connect('localhost', 'root', '', 'tests')) {
      mysqli_set_charset($db, 'utf8');
      $query = 'UPDATE `counter` SET `total` = `total` + 1 ';
      $query .= 'WHERE `id_link`=' . $id;
      @mysqli_query($db, $query);
      mysqli_close($db);
   }
}
header('Location: ' . $_GET['url']);
exit();
```

Оператор `+` позволяет увеличить счетчик за один запрос. Иначе пришлось бы вначале получить значение из базы данных, затем увеличить его и только во втором запросе обновить значение в базе данных.

6.7.2. Побитовые операторы

Приведем список побитовых операторов:

- `~` — двоичная инверсия;
- `&` — двоичное `И`:

  ```
  SELECT BIN(100 & 75);
  /* Выведет: 1000000 */
  ```

- `|` — двоичное `ИЛИ`:

```
SELECT BIN(100 | 75);
/* Выведет: 1101111 */
```

- `^` — двоичное исключающее `ИЛИ`:

```
SELECT BIN(100 ^ 250);
/* Выведет: 10011110 */
```

- `<<` — сдвиг влево на один или более разрядов с заполнением младших разрядов нулями:

```
SELECT BIN(100 << 1);
/* Выведет: 11001000 */
```

- `>>` — сдвиг вправо на один или более разрядов с заполнением старших разрядов содержимым самого старшего разряда:

```
SELECT BIN(100 >> 1);
/* Выведет: 110010 */
SELECT BIN(-127 >> 1);
/* Выведет:
1111111111111111111111111111111111111111111111111111111111000000
*/
```

6.7.3. Операторы сравнения

Операторы сравнения используются, прежде всего, в конструкциях `WHERE` и `HAVING` при создании запросов. Приведем их список:

- `=` — равно;
- `<=>` — эквивалентно;
- `!=` — не равно;
- `<>` — не равно;
- `<` — меньше;
- `>` — больше;
- `<=` — меньше или равно;
- `>=` — больше или равно;
- `IS NOT NULL` — проверка на наличие значения;
- `IS NULL` — проверка поля на отсутствие значения;
- `BETWEEN <Начало> AND <Конец>` — проверяет, является ли значение большим или равным `<Начало>` и меньшим или равным `<Конец>`:

```
MariaDB [tests]> SELECT `Name` FROM `Customers`
    -> WHERE `id_Customer` BETWEEN 1 AND 2;
```

```
+--------------------------+
| Name                     |
+--------------------------+
| Иванов Иван Иванович     |
| Петров Сергей Николаевич |
+--------------------------+
```

- `IN` — содержится в определенном наборе:

```
MariaDB [tests]> SELECT `id_Product` FROM `Products`
    -> WHERE `Product` IN ('Монитор', 'HDD');
+------------+
| id_Product |
+------------+
|          1 |
|          3 |
+------------+
```

- `NOT IN` — не содержится в определенном наборе:

```
WHERE `Product` NOT IN ('Монитор', 'HDD');
```

- `LIKE` — соответствие шаблону SQL:

```
MariaDB [tests]> SELECT `Product` FROM `Products`
    -> WHERE `Product` LIKE 'Р%';
+---------+
| Product |
+---------+
| Ручка   |
+---------+
```

- `NOT LIKE` — несоответствие шаблону SQL;
- `REGEXP` — соответствие регулярному выражению:

```
MariaDB [tests]> SELECT `Product` FROM `Products`
    -> WHERE `Product` REGEXP '^p+';
+---------+
| Product |
+---------+
| Ручка   |
+---------+
```

- `RLIKE` — соответствие регулярному выражению (синоним `REGEXP`);
- `NOT REGEXP` — несоответствие регулярному выражению;
- `NOT RLIKE` — несоответствие регулярному выражению (синоним `NOT REGEXP`).

В шаблоне SQL могут использоваться следующие символы:

- `%` — любое число символов;
- `_` — любой одиночный символ.

Можно проверять сразу несколько условий, указав логические операции:

- AND или && — логическое И;
- OR или || — логическое ИЛИ.

Результаты операции сравнения:

- 0 — ложь;
- 1 — истина;
- NULL — возвращается, если хотя бы один из операндов равен NULL.

Исключением является оператор эквивалентности <=>. Он возвращает только два значения: 0 (ложь) и 1 (истина). Этот оператор введен специально для сравнения значения NULL.

Следует отметить, что по умолчанию сравнение строк происходит без учета регистра. Если указать ключевое слово BINARY, то регистр символов будет учитываться:

```
SELECT 'TEXT'='text';
/* Выведет: 1 (истина) */
SELECT BINARY 'TEXT'='text';
/* Выведет: 0 (ложь) */
```

Результат сравнения можно изменить на противоположный с помощью операторов ! и NOT:

```
SELECT 'TEXT'='text';
/* Выведет: 1 (истина) */
SELECT !('TEXT'='text');
/* Выведет: 0 (ложь) */
```

Логические выражения следует заключать в круглые скобки, т. к. приоритет оператора отрицания ! выше приоритета других операторов.

6.7.4. Операторы присваивания

Приведем список операторов присваивания:

- := — сохраняет значение в переменной SQL:

  ```
  SELECT @time := NOW();
  ```

- = — сохраняет значение в переменной SQL. Можно использовать в случае применения оператора SET:

  ```
  SET @time = NOW();
  ```

6.7.5. Приоритет выполнения операторов

При составлении выражений следует учитывать приоритет выполнения операторов.

Приведем список операторов в порядке убывания их приоритета:

- BINARY, COLLATE.
- !.

- `-` (унарный минус), `~`.
- `^`.
- `*`, `/`, `%`, `MOD`, `DIV`.
- `+`, `-` — сложение, вычитание.
- `<<`, `>>` — двоичные сдвиги.
- `&` — двоичное `И`.
- `|` — двоичное `ИЛИ`.
- `=` (равно), `<=>`, `>=`, `<=`, `>`, `<`, `<>`, `!=`, `IS`, `LIKE`, `REGEXP`, `IN`.
- `BETWEEN`.
- `NOT`.
- `&&`, `AND`.
- `XOR`.
- `||`, `OR`.
- `=` (присваивание), `:=`.

С помощью круглых скобок можно изменить последовательность выполнения выражения:

```
SELECT 5 + 3 * 7;
/* Выведет: 26 */
SELECT (5 + 3) * 7;
/* Выведет: 56 */
```

6.7.6. Преобразование типов данных

В большинстве случаев преобразование типов осуществляется автоматически. В этом разделе мы рассмотрим результаты автоматического преобразования типов, а также встроенные функции для специального приведения типов.

Что будет, если к числу прибавить строку?

```
SELECT '5' + 3;
/* Выведет: 8 */
SELECT '5st' + 3;
/* Выведет: 8 */
```

В этом случае строка преобразуется в число, а затем выполняется операция сложения. Но что будет, если строку невозможно преобразовать в число?

```
SELECT 'str' + 3;
/* Выведет: 3 */
SELECT 3 + 'str';
/* Выведет: 3 */
```

Если строку невозможно преобразовать в число, то она приравнивается к нулю.

Для явного преобразования типов предназначены две функции:

```
CAST(<Выражение> AS <Тип>)
CONVERT(<Выражение>, <Тип>)
CONVERT(<Выражение> USING <Кодировка>)
```

Параметр `<Тип>` может принимать следующие значения:

- `BINARY`;
- `CHAR`;
- `DATE`;
- `DATETIME`;
- `DECIMAL`;
- `SIGNED [INTEGER]`;
- `TIME`;
- `UNSIGNED [INTEGER]`.

Пример:

```
SELECT CAST('2017-11-04' AS DATETIME);
/* Выведет: 2017-11-04 00:00:00 */
SELECT CONVERT('2017-11-04', DATETIME);
/* Выведет: 2017-11-04 00:00:00 */
```

6.8. Поиск по шаблону

Для поиска по шаблону предусмотрены два оператора:

- `LIKE` — соответствие шаблону SQL;
- `NOT LIKE` — несоответствие шаблону SQL.

В шаблоне SQL могут присутствовать следующие специальные символы:

- `%` — любое число символов;
- `_` — любой одиночный символ.

Если специальные символы не используются, то применение оператора `LIKE` эквивалентно оператору `=`:

```
SELECT 'строка для поиска' LIKE 'поиск';
/* Выведет: 0 */
SELECT 'поиск' LIKE 'поиск';
/* Выведет: 1 */
```

Следует помнить, что при поиске в текстовом поле регистр не учитывается. Чтобы учитывался регистр, необходимо указать ключевое слово `BINARY`:

```
SELECT 'PHP' LIKE 'php';
/* Выведет: 1 */
```

```
SELECT 'PHP' LIKE BINARY 'php';
/* Выведет: 0 */
```

Специальные символы могут быть расположены в любом месте шаблона. Например, чтобы найти все вхождения, необходимо указать символ `%` в начале и в конце шаблона:

```
SELECT 'строка для поиска' LIKE '%поиск%';
/* Выведет: 1 */
```

Можно установить привязку или только к началу строки, или только к концу:

```
SELECT 'строка для поиска' LIKE 'строка%';
/* Выведет: 1 */
SELECT 'новая строка для поиска' LIKE 'строка%';
/* Выведет: 0 */
SELECT 'строка для поиска' LIKE '%поиска';
/* Выведет: 1 */
SELECT 'строка для поиска 2' LIKE '%поиска';
/* Выведет: 0 */
```

Шаблон для поиска может иметь очень сложную структуру:

```
SELECT 'строка для поиска' LIKE '%строк_%по_ск%';
/* Выведет: 1 */
SELECT 'строка для поиска' LIKE '%поиск%а';
/* Выведет: 1 */
```

Обратите внимание на последнюю строку поиска. Этот пример демонстрирует, что специальный символ `%` соответствует не только любому числу символов, но и полному их отсутствию.

Что же делать, если необходимо найти символы `%` и _? Ведь они являются специальными:

```
SELECT 'скидка 10%' LIKE '%10%';
/* Выведет: 1 */
SELECT 'скидка 10$' LIKE '%10%';
/* Выведет: 1 */
```

В этом случае специальные символы необходимо экранировать с помощью обратной косой черты:

```
SELECT 'скидка 10%' LIKE '%10\%';
/* Выведет: 1 */
SELECT 'скидка 10$' LIKE '%10\%';
/* Выведет: 0 */
```

Обратите внимание, что функция `mysqli_real_escape_string()` не добавляет обратную косую черту перед символами `%` и _ для их защиты. Сделать это в PHP позволяет функция `addcslashes()`:

```
$text = addcslashes($text, '_%');
```

В качестве примера рассмотрим поиск по шаблону. Для этого создадим таблицу `search` в базе данных `tests`:

```
CREATE TABLE `search` (
   `id` INT NOT NULL AUTO_INCREMENT,
   `str` TEXT,
   PRIMARY KEY (`id`)
) ENGINE=MyISAM CHARSET=utf8 COLLATE utf8_general_ci;
```

Затем добавим две записи:

```
INSERT INTO `search` VALUES (NULL, 'Скидка 10%');
INSERT INTO `search` VALUES (NULL, 'Скидка 10$');
```

Исходный код с вариантами поиска по шаблону приведен в листинге 6.5.

Листинг 6.5. Поиск по шаблону

```
$str_search = '10%';
if (@$db = mysqli_connect('localhost', 'root', '', 'tests')) {
   mysqli_set_charset($db, 'utf8');
   // Добавляем защитные слеши
   $str_search = mysqli_real_escape_string($db, $str_search);

   echo 'Без добавления слешей перед спецсимволами:<br>';
   $query = "SELECT `str` FROM `search` WHERE `str` LIKE '%";
   $query .= $str_search . "%'";
   if ($res = mysqli_query($db, $query)) {
      while ($row = mysqli_fetch_row($res)) {
         echo $row[0] . '<br>';
      }
      mysqli_free_result($res);
   }

   echo '<br>После добавления слешей перед спецсимволами:<br>';
   $str_search = addcslashes($str_search, '_%');
   $query = "SELECT `str` FROM `search` WHERE `str` LIKE '%";
   $query .= $str_search . "%'";
   if ($res = mysqli_query($db, $query)) {
      while ($row = mysqli_fetch_row($res)) {
         echo $row[0] . '<br>';
      }
      mysqli_free_result($res);
   }
   mysqli_close($db);
}
else {
   echo 'Не удалось установить подключение к базе данных';
}
```

Открыв этот файл в Web-браузере, мы увидим:

```
Без добавления слешей перед спецсимволами:
Скидка 10%
Скидка 10$

После добавления слешей перед спецсимволами:
Скидка 10%
```

В этом примере предполагается, что значение переменной `$str_search` было получено через форму поиска. Поэтому, прежде чем подставить значение переменной в SQL-запрос, мы экранируем специальные символы. Если этого не сделать, то любой пользователь сможет видоизменить SQL-запрос.

6.9. Поиск с помощью регулярных выражений

Регулярные выражения дают возможность осуществить сложный поиск. Использовать регулярные выражения позволяют следующие операторы:

- `REGEXP` — соответствие регулярному выражению;
- `RLIKE` — соответствие регулярному выражению (синоним `REGEXP`);
- `NOT REGEXP` — несоответствие регулярному выражению;
- `NOT RLIKE` — несоответствие регулярному выражению (синоним `NOT REGEXP`).

При использовании регулярных выражений следует помнить, что такой поиск выполняется медленнее, чем поиск по шаблону, и отнимает у сервера больше системных ресурсов. Кроме того, при работе с многобайтовыми кодировками регулярные выражения могут давать некорректный результат.

Синтаксис регулярных выражений MySQL, PHP и JavaScript схож. В регулярных выражениях может встречаться ряд метасимволов:

- `^` — привязка к началу строки.
- `$` — привязка к концу строки.

Привязки работают так:

```
SELECT '2' REGEXP '^[0-9]+$';
/* Выведет: 1 */
SELECT 'Строка2' REGEXP '^[0-9]+$';
/* Выведет: 0 */
```

Если убрать привязку к началу и концу строки, то любая строка, содержащая цифру, вернет `1`:

```
SELECT 'Строка2' REGEXP '[0-9]+';
/* Выведет: 1 */
```

Можно указать привязку только к началу или только к концу строки:

```
SELECT 'Строка2' REGEXP '[0-9]+$';
/* Есть цифра в конце строки, значит выведет: 1 */
```

```
SELECT 'Строка2' REGEXP '^[0-9]+';
/* Нет цифры в начале строки, значит выведет: 0 */
```

Регулярное выражение `'^$'` соответствует пустой строке. Если необходимо найти значение `NULL`, то следует использовать не регулярные выражения, а операторы `IS NULL` и `IS NOT NULL`.

- `[[:<:]]` — привязка к началу слова.
- `[[:>:]]` — привязка к концу слова.

```
SELECT 'в середине строки' REGEXP '[[:<:]]середине[[:>:]]';
/* Выведет: 1 */
```

Квадратные скобки `[]` позволяют указать символы, которые могут встречаться на этом месте в строке. Можно записать символы подряд или указать диапазон через дефис:

- `[09]` — цифра `0` или `9`;
- `[0-9]` — любая цифра от `0` до `9`;
- `[абв]` — буквы `а`, `б` или `в`;
- `[а-г]` — буквы `а`, `б`, `в` или `г`;
- `[а-яё]` — любая русская буква от `а` до `я`;
- `[0-9а-яёa-z]` — любая цифра и любая русская или латинская буква.

Значение можно инвертировать, если после первой скобки указать символ `^`. Таким способом можно указать символы, которых не должно быть на этом месте в строке:

- `[^09]` — не цифра `0` и не цифра `9`;
- `[^0-9]` — не цифра от `0` до `9`;
- `[^а-яёa-z]` — не русская или латинская буква.

Поиск выполняется без учета регистра символов. Чтобы учитывался регистр, необходимо указать ключевое слово `BINARY`:

```
SELECT 'СтрОКа' REGEXP '^[а-яё]+$';
/* Выведет: 1 */
SELECT 'СтрОКа' REGEXP BINARY '^[а-яё]+$';
/* Выведет: 0 */
```

Если при использовании кодировки UTF-8 мы укажем ключевое слово `BINARY`, то не сможем получить соответствие, даже если строка состоит только из строчных букв:

```
SELECT 'строка' REGEXP BINARY '^[а-яё]+$';
/* При использовании cp1251 выведет: 1 */
SELECT 'строка' REGEXP BINARY '^[а-яё]+$';
/* При использовании utf8 выведет: 0 */
```

Вместо перечисления символов можно использовать стандартные классы:

- `[[:alnum:]]` — алфавитно-цифровые символы;
- `[[:alpha:]]` — буквенные символы;

- `[[:lower:]]` — строчные буквы;
- `[[:upper:]]` — прописные буквы;
- `[[:digit:]]` — десятичные цифры;
- `[[:xdigit:]]` — шестнадцатеричные цифры;
- `[[:punct:]]` — знаки пунктуации;
- `[[:blank:]]` — символы табуляции и пробелов;
- `[[:space:]]` — символы пробела, табуляции, новой строки или возврата каретки;
- `[[:cntrl:]]` — управляющие символы;
- `[[:print:]]` — печатные символы;
- `[[:graph:]]` — печатные символы, за исключением пробельных;
- `.` (точка) — любой символ.

Что же делать, если нужно найти точку, ведь символ «точка» соответствует любому символу? Для этого перед специальным символом необходимо указать два символа `\`, т. е. так: `\\.`.

```
SELECT '26,11.2017'
REGEXP '^[0-3][0-9].[01][0-9].[12][09][0-9][0-9]$';
/* Поскольку точка означает любой символ, выведет: 1 */
SELECT '26,11.2017'
REGEXP '^[0-3][0-9]\\.[01][0-9]\\.[12][09][0-9][0-9]$';
/* Поскольку перед точкой указаны символы \\, выведет: 0 */
```

Число вхождений предшествующего символа или выражения в строку задается с помощью *квантификаторов*:

- `{n}` — `n` вхождений символа (выражения) в строку:

  `[0-9]{2}` — соответствует двум вхождениям любой цифры;
- `{n,}` — `n` или более вхождений символа в строку:

  `[0-9]{2,}` — соответствует двум и более вхождениям любой цифры;
- `{n,m}` — не менее `n` и не более `m` вхождений символа в строку. Цифры указываются через запятую без пробела:

  `[0-9]{2,5}` — соответствует двум, трем, четырем или пяти вхождениям любой цифры;
- `*` — любое число вхождений символа в строку, в том числе ни одного:

  `[0-9]*` — цифры могут не встретиться в строке или встретиться много раз;
- `+` — как минимум одно вхождение символа в строку:

  `[0-9]+` — цифра может встретиться один или много раз;
- `?` — одно или ни одного вхождения символа в строку:

  `[0-9]?` — цифра может встретиться один раз или не встретиться совсем.

Для указания числа вхождений нескольких символов используются круглые скобки:

```
SELECT '121212' REGEXP '^(12){3}$';
/* Выведет: 1 */
```

Также можно искать одно из двух выражений: `n|m`, где `n` или `m` — один из символов (одно из выражений):

`красн(ая)|(ое)` — `красная` или `красное`, но не красный:

```
SELECT 'красная' REGEXP 'красн(ая)|(ое)';
/* Выведет: 1 */
```

6.10. Режим полнотекстового поиска

Кроме поиска по шаблону и применения регулярных выражений, для таблиц типа `MyISAM` или `InnoDB` можно задать режим *полнотекстового поиска*. Столбцы, используемые для поиска, должны быть проиндексированы с помощью специального индекса `FULLTEXT`. Индексации подлежат столбцы, имеющие тип `CHAR`, `VARCHAR` или `TEXT`.

6.10.1. Создание индекса *FULLTEXT*

Создать индекс `FULLTEXT` можно следующими способами:

- при создании таблицы посредством оператора `CREATE TABLE` с помощью инструкции:

  ```
  FULLTEXT INDEX <Название индекса> (<Столбцы через запятую>)
  ```

  Например, так:

  ```
  CREATE TABLE `search1` (
     `id` INT NOT NULL AUTO_INCREMENT,
     `str` TEXT,
     FULLTEXT INDEX `index1` (`str`),
     PRIMARY KEY (`id`)
  ) ENGINE=MyISAM CHARSET=utf8 COLLATE utf8_general_ci;
  ```

  или, если требуется создать индекс на основе сразу нескольких полей:

  ```
  CREATE TABLE `search2` (
     `id` INT NOT NULL AUTO_INCREMENT,
     `str1` TEXT,
     `str2` TEXT,
     FULLTEXT INDEX `index2` (`str1`, `str2`),
     PRIMARY KEY (`id`)
  ) ENGINE=MyISAM CHARSET=utf8 COLLATE utf8_general_ci;
  ```

- с помощью оператора `ALTER TABLE`.

  Например, создав таблицу:

  ```
  CREATE TABLE `search3` (
     `id` INT NOT NULL AUTO_INCREMENT,
  ```

```
    `str` TEXT,
    PRIMARY KEY (`id`)
) ENGINE=MyISAM CHARSET=utf8 COLLATE utf8_general_ci;
```

можно добавить к ней полнотекстовый индекс так:

```
ALTER TABLE `search3` ADD FULLTEXT `index3` (`str`);
```

А к таблице с двумя текстовыми полями:

```
CREATE TABLE `search4` (
    `id` INT NOT NULL AUTO_INCREMENT,
    `str1` TEXT,
    `str2` TEXT,
    PRIMARY KEY (`id`)
) ENGINE=MyISAM CHARSET=utf8 COLLATE utf8_general_ci;
```

можно добавить индекс так:

```
ALTER TABLE `search4` ADD FULLTEXT `index4` (`str1`, `str2`);
```

- с помощью оператора `CREATE INDEX`.

  Например, к созданной ранее таблице с одним текстовым полем можно добавить индекс так:

```
CREATE FULLTEXT INDEX `index5` ON `search3` (`str`);
```

  А к таблице с двумя полями так:

```
CREATE FULLTEXT INDEX `index6` ON `search4` (`str1`, `str2`);
```

При использовании таблиц `MyISAM` в индекс попадут слова длиной от 4 до 84 символов. Эти значения задаются переменными `ft_min_word_len` и `ft_max_word_len` соответственно. Изменить значения этих переменных можно через конфигурационный файл my.ini. После изменения значения переменных необходимо заново создать индекс `FULLTEXT`. Посмотреть текущие значения переменных позволяет SQL-команда:

```
SHOW VARIABLES LIKE 'ft%';
```

При использовании таблиц `InnoDB` в индекс попадут слова длиной от 3 до 84 символов. Эти значения задаются переменными `innodb_ft_min_token_size` и `innodb_ft_max_token_size` соответственно. Посмотреть текущие значения переменных позволяет SQL-команда:

```
SHOW VARIABLES LIKE 'innodb_ft%';
```

Следует отметить, что полнотекстовый поиск предназначен для поиска в большом объеме текста. Если содержимое поля состоит из нескольких слов, то оно может вообще не попасть в индекс.

6.10.2. Реализация полнотекстового поиска

Полнотекстовый поиск выполняется с помощью конструкции `MATCH(...) AGAINST(...)`, которая имеет следующий формат:

```
MATCH(<Поля через запятую>)
AGAINST('<Строка для поиска>' [<Модификатор>])
```

Необязательный параметр `<Модификатор>` может принимать следующие значения:

- `IN NATURAL LANGUAGE MODE` — значение по умолчанию;
- `IN BOOLEAN MODE` — режим логического поиска;
- `WITH QUERY EXPANSION` — поиск с расширением запроса.

Для примера добавим три записи в таблицу `search1`:

```
INSERT INTO `search1` VALUES (NULL, 'При использовании таблиц MyISAM в индекс
попадут слова длиной от 4 до 84 символов. Данные значения задаются переменными
ft_min_word_len и ft_max_word_len соответственно. Изменить значения этих
переменных можно через конфигурационный файл my.ini. После изменения значения
переменных необходимо заново создать индексы FULLTEXT.');
INSERT INTO `search1` VALUES (NULL, 'Запись 2');
INSERT INTO `search1` VALUES (NULL, 'Строка 3');
INSERT INTO `search1` VALUES (NULL, 'база данных MySQL');
INSERT INTO `search1` VALUES (NULL, 'MySQL');
```

> ***Внимание!***
> Для реализации полнотекстового поиска в таблице должно быть не менее трех записей.

А теперь найдем строку с помощью полнотекстового поиска:

```
SELECT * FROM `search1` WHERE MATCH(`str`) AGAINST('значения');
```

Результаты поиска сортируются по коэффициенту релевантности, который представляет собой число с плавающей точкой. Чтобы увидеть этот коэффициент, воспользуемся следующим SQL-запросом:

```
MariaDB [tests]> SELECT MATCH(`str`) AGAINST('файл') AS iq
    -> FROM `search1`;
+--------------------+
| iq                 |
+--------------------+
| 0.9790974855422974 |
|                  0 |
|                  0 |
|                  0 |
|                  0 |
+--------------------+
```

6.10.3. Режим логического поиска

Режим логического поиска позволяет использовать специальные символы, которые влияют на значение коэффициента релевантности. Чтобы применить режим логического поиска, необходимо в конструкции `MATCH(...) AGAINST(...)` указать модификатор `IN BOOLEAN MODE`.

Приведем специальные символы логического режима:

- `+` — слово обязательно должно присутствовать в результате:

```
SELECT * FROM `search1`
WHERE MATCH(`str`)
AGAINST('конфигурационный +файл' IN BOOLEAN MODE);
```

- `-` — слово не должно присутствовать в результате:

```
SELECT * FROM `search1`
WHERE MATCH(`str`)
AGAINST('конфигурационный -файл' IN BOOLEAN MODE);
```

- `<` — уменьшает вклад слова в коэффициент релевантности:

```
SELECT * FROM `search1`
WHERE MATCH(`str`)
AGAINST('конфигурационный <файл' IN BOOLEAN MODE);
```

- `>` — увеличивает вклад слова в коэффициент релевантности:

```
SELECT * FROM `search1`
WHERE MATCH(`str`)
AGAINST('конфигурационный >файл' IN BOOLEAN MODE);
```

- `()` — круглые скобки служат для группировки слов в подвыражения;
- `~` — символ для указания нежелательного слова. В отличие от символа `-`, символ `~` не исключает слово из результата, а лишь уменьшает коэффициент релевантности;
- `*` — символ усечения. Указывается в конце слова;
- `""` — строка должна содержать точную фразу:

```
SELECT * FROM `search1`
WHERE MATCH(`str`)
AGAINST('"конфигурационный файл"' IN BOOLEAN MODE);
```

6.10.4. Поиск с расширением запроса

При поиске с расширением запроса поиск фактически выполняется дважды. При первом поиске отбираются все записи, содержащие искомые слова. А при выполнении второго поиска ищутся записи, что включают наиболее релевантные слова из записей, найденных при первом поиске, даже если самих искомых слов в них нет. Все записи, найденные в процессе выполнения обоих поисков, объединяются и возвращаются в качестве результата.

В качестве примера можно рассмотреть поиск записей, содержащих словосочетание `база данных`. При первом поиске будет найдена запись, которая содержит словосочетание `база данных MySQL`. Тогда при втором поиске будут отобраны все записи, содержащие слово `MySQL`. Результатом такого поиска будет набор, включающий как записи со словосочетанием `база данных`, так и записи, что содержат слово `MySQL`.

Чтобы применить режим поиска с расширением запроса, необходимо в конструкции `MATCH(...) AGAINST(...)` указать модификатор `WITH QUERY EXPANSION`:

```
MariaDB [tests]> SELECT * FROM `search1`
    -> WHERE MATCH(`str`)
    -> AGAINST('база данных' WITH QUERY EXPANSION);
+----+-------------------+
| id | str               |
+----+-------------------+
|  4 | база данных MySQL |
|  5 | MySQL             |
+----+-------------------+
```

6.11. Функции MySQL

MySQL имеет множество встроенных *функций*, которые позволяют выполнять с данными определенные действия. Например, функция `NOW()` возвращает текущие дату и время, а функция `DATE_FORMAT()` преобразует формат вывода даты.

При использовании функций следует помнить, что между круглыми скобками и именем функции не должно быть пробела, а скобки нужно обязательно указывать, даже если в функцию не передаются аргументы. Рассмотрим функции MySQL более подробно.

> **Примечание**
> Агрегатные функции мы рассматривали в *разд. 6.4.8*.

6.11.1. Функции для работы с числами

Стандартные тригонометрические функции (аргументы должны задаваться в радианах):

- `SIN()` — синус;
- `COS()` — косинус;
- `TAN()` — тангенс;
- `COT()` — котангенс.

Обратные тригонометрические функции (возвращают значение в радианах):

- `ASIN()` — арксинус;
- `ACOS()` — арккосинус;
- `ATAN()` — арктангенс.

Округление чисел:

- `CEILING()` — значение, округленное до ближайшего большего целого:

  ```
  SELECT CEILING(4.3);
  /* Выведет: 5 */
  ```

- `CEIL()` — то же, что и `CEILING()`:

```
SELECT CEIL(4.3);
/* Выведет: 5 */
```

- `FLOOR()` — значение, округленное до ближайшего меньшего целого:

```
SELECT FLOOR(4.5);
/* Выведет: 4 */
```

- `ROUND(<X>[, <Y>])` — значение, округленное до ближайшего меньшего целого для чисел с дробной частью, меньшей `0.5`, или до ближайшего большего целого для чисел с дробной частью, равной или большей `0.5`:

```
SELECT ROUND(4.49);
/* Выведет: 4 */
SELECT ROUND(4.5);
/* Выведет: 5 */
SELECT ROUND(4.51);
/* Выведет: 5 */
```

  Вторым аргументом для функции можно указать число знаков после запятой, до которых нужно округлить число:

```
SELECT ROUND(4.49, 1);
/* Выведет: 4.5 */
SELECT ROUND(12.34321, 3);
/* Выведет: 12.343 */
```

- `TRUNCATE(<X>, <Y>)` — возвращает дробное число `<X>`, имеющее `<Y>` знаков после запятой. Если в качестве значения аргумента `<Y>` передать значение `0`, то функция вернет число, округленное до меньшего целого:

```
SELECT TRUNCATE(4.55, 0);
/* Выведет: 4 */
SELECT TRUNCATE(4.55, 1);
/* Выведет: 4.5 */
SELECT TRUNCATE(4.55, 3);
/* Выведет: 4.550 */
```

Функции для преобразования чисел:

- `CONV(<Число>, <Исходная система>, <Нужная система>)` — преобразует число из одной системы счисления в другую:

```
SELECT CONV(255, 10, 16);
/* Выведет: FF */
SELECT CONV('FF', 16, 10);
/* Выведет: 255 */
```

- `BIN(<Число>)` — преобразует число из десятичной системы счисления в двоичную:

```
SELECT BIN(17);
/* Выведет: 10001 */
```

- `HEX(<Число>)` — возвращает значение аргумента в виде шестнадцатеричного числа:

```
SELECT HEX(255);
/* Выведет: FF */
```

- `OCT(<Число>)` — преобразует число из десятичной системы счисления в восьмеричную:

```
SELECT OCT(10);
/* Выведет: 12 */
```

Прочие функции:

- `ABS()` — абсолютное значение:

```
SELECT ABS(-4.55);
/* Выведет: 4.55 */
```

- `EXP()` — экспонента;
- `LOG(<X>)` — натуральный логарифм;
- `LOG2(<X>)` — логарифм числа по основанию `2`:

```
SELECT LOG2(128);
/* Выведет: 7 */
```

- `LOG10(<X>)` — логарифм числа по основанию `10`:

```
SELECT LOG10(100);
/* Выведет: 2 */
```

- `LOG(<Основание>, <X>)` — логарифм числа `<X>` по основанию `<Основание>`:

```
SELECT LOG(2, 128);
/* Выведет: 7 */
SELECT LOG(10, 100);
/* Выведет: 2 */
```

- `POW(<Число>, <Степень>)` — возводит `<Число>` в `<Степень>`:

```
SELECT POW(5, 2);
/* Выведет: 25 */
```

- `SQRT()` — извлекает квадратный корень:

```
SELECT SQRT(25);
/* Выведет: 5 */
```

- `PI()` — возвращает число π:

```
SELECT PI();
/* Выведет: 3.141593 */
```

- `MOD(<X>, <Y>)` — определяет остаток от деления `<X>` на `<Y>`:

```
SELECT MOD(10, 2);
/* Выведет: 0 */
```

- `DEGREES()` — преобразует значение угла из радиан в градусы:

  ```
  SELECT DEGREES(PI());
  /* Выведет: 180 */
  ```

- `RADIANS()` — преобразует значение угла из градусов в радианы:

  ```
  SELECT RADIANS(180);
  /* Выведет: 3.141592653589793 */
  ```

- `SIGN()` — возвращает `-1`, если число отрицательное, `1`, если число положительное, и `0`, если число равно нулю:

  ```
  SELECT SIGN(-80);
  /* Выведет: -1 */
  SELECT SIGN(80);
  /* Выведет: 1 */
  ```

- `LEAST()` — служит для определения минимального значения из списка:

  ```
  SELECT LEAST(2, 1, 3);
  /* Выведет: 1 */
  ```

- `GREATEST()` — позволяет определить максимальное значение из списка:

  ```
  SELECT GREATEST(2, 1, 3);
  /* Выведет: 3 */
  ```

- `FORMAT(<Число>, <Число знаков после запятой>[, <Локаль>])` — форматирует число в строку с заданным числом знаков после запятой в соответствии с заданной локалью. Если локаль не указана, используется американская (`en_US`):

  ```
  SELECT FORMAT(56873.8732, 2);
  /* Выведет: 56,873.87 */
  SELECT FORMAT(56873.8732, 2, 'ru_RU');
  /* Здесь мы указали российскую локаль */
  /* Выведет: 56 873,87 */
  ```

- `RAND()` — возвращает случайное число в диапазоне от `0` до `1`. Если в функцию передать параметр, то это настроит генератор на новую последовательность. Следует учитывать, что при передаче одного и того же параметра функция выдает одну и ту же последовательность:

  ```
  SELECT RAND();
  /* Выведет: 0.35286363153985106 */
  SELECT RAND();
  /* Выведет: 0.7805252687824195 */
  SELECT RAND(10);
  /* Выведет: 0.6570515219653505 */
  SELECT RAND(10);
  /* Выведет: 0.6570515219653505 */
  ```

В качестве примера рассмотрим вывод записи из базы данных случайным образом. Предположим, что наш сайт — развлекательный портал, и в базе данных есть таб-

лица, заполненная анекдотами. При каждом запросе страницы мы будем выводить один анекдот случайным образом. Для этого в базе данных `tests` создадим таблицу `anecdotes`:

```
CREATE TABLE `anecdotes` (
   `id` INT NOT NULL AUTO_INCREMENT,
   `anecdote` TEXT,
   PRIMARY KEY (`id`)
) ENGINE=MyISAM CHARSET=utf8 COLLATE utf8_general_ci;
```

Затем добавим несколько записей:

```
INSERT INTO `anecdotes` VALUES (NULL, 'Анекдот 1');
INSERT INTO `anecdotes` VALUES (NULL, 'Анекдот 2');
INSERT INTO `anecdotes` VALUES (NULL, 'Анекдот 3');
INSERT INTO `anecdotes` VALUES (NULL, 'Анекдот 4');
INSERT INTO `anecdotes` VALUES (NULL, 'Анекдот 5');
```

Исходный код для вывода анекдота случайным образом приведен в листинге 6.6.

Листинг 6.6. Вывод анекдота случайным образом

```
if (@$db = mysqli_connect('localhost', 'root', '', 'tests')) {
   mysqli_set_charset($db, 'utf8');
   $query = 'SELECT * FROM `anecdotes` ORDER BY RAND() LIMIT 1';
   if ($res = mysqli_query($db, $query)) {
      if ($row = mysqli_fetch_row($res)) {
         echo $row[1] . '<br>';
      }
      mysqli_free_result($res);
   }
   mysqli_close($db);
}
else {
   echo 'Не удалось установить подключение к базе данных';
}
```

6.11.2. Функции даты и времени

Для получения текущих даты и времени предусмотрены следующие функции:

- `NOW()`, `LOCALTIME()` и `LOCALTIMESTAMP()` — возвращают текущие дату и время для временной зоны, установленной в системных настройках, в формате `ГГГГ-ММ-ДД ЧЧ:ММ:СС`:

```
SELECT NOW();
/* Выведет: 2018-02-12 05:59:43 */
SELECT LOCALTIME();
/* Выведет: 2018-02-12 05:59:43 */
SELECT LOCALTIMESTAMP();
/* Выведет: 2018-02-12 05:59:43 */
```

Выведем информацию об установленной временной зоне:

```
SELECT @@time_zone;
/* Выведет: SYSTEM */
```

Укажем временную зону явным образом:

```
SET time_zone = '+03:00';
SELECT @@time_zone;
/* Выведет: +03:00 */
```

- `UTC_TIMESTAMP()` — выводит текущие дату и время по Гринвичу в формате `ГГГГ-ММ-ДД ЧЧ:ММ:СС`:

```
SELECT UTC_TIMESTAMP();
/* Выведет: 2018-02-12 03:00:57 */
```

- `SYSDATE()` — позволяет определить текущие дату и время в формате `ГГГГ-ММ-ДД ЧЧ:ММ:СС`:

```
SELECT SYSDATE();
/* Выведет: 2018-02-12 06:03:22 */
```

В отличие от функции `NOW()` и ее синонимов, `SYSDATE()` возвращает время, в которое она была вызвана, тогда как `NOW()` возвращает время начала выполнения запроса;

- `CURDATE()` и `CURRENT_DATE()` — возвращают текущую дату для временной зоны, установленной в системных настройках, в формате `ГГГГ-ММ-ДД`:

```
SELECT CURDATE();
/* Выведет: 2018-02-12 */
SELECT CURRENT_DATE();
/* Выведет: 2018-02-12 */
```

- `UTC_DATE()` — позволяет определить текущую дату по Гринвичу в формате `ГГГГ-ММ-ДД`:

```
SELECT UTC_DATE();
/* Выведет: 2018-02-12 */
```

- `CURTIME()` и `CURRENT_TIME()` — возвращают текущее время для временной зоны, установленной в системных настройках, в формате `ЧЧ:ММ:СС`:

```
SELECT CURTIME();
/* Выведет: 06:14:18 */
SELECT CURRENT_TIME();
/* Выведет: 06:14:18 */
```

- `UTC_TIME()` — сообщает текущее время по Гринвичу в формате `ЧЧ:ММ:СС`:

```
SELECT UTC_TIME();
/* Выведет: 03:17:25 */
```

- `UNIX_TIMESTAMP()` — подсчитывает число секунд, прошедших с полуночи 1 января 1970 г.:

```
SELECT UNIX_TIMESTAMP();
/* Выведет: 1518451863 */
```

Ряд функций позволяют получить следующие фрагменты даты и времени:

- DATE() — дата:

```
SELECT DATE('2018-02-12 06:03:22');
/* Выведет: 2018-02-12 */
```

- YEAR() — год:

```
SELECT YEAR('2018-02-12 06:03:22');
/* Выведет: 2018 */
```

- MONTH() — месяц:

```
SELECT MONTH('2018-02-12 06:03:22');
/* Выведет: 2 */
```

- MONTHNAME() — название месяца в виде строки в зависимости от настроек локали (по умолчанию используется английская локаль en_US):

```
SELECT @@lc_time_names;
/* Выведет: en_US */
SELECT MONTHNAME('2018-02-12 06:03:22');
/* Выведет: February */
```

- DAY() и DAYOFMONTH() — номер дня в месяце:

```
SELECT DAY('2018-02-12 06:03:22');
/* Выведет: 12 */
SELECT DAYOFMONTH('2018-02-12 06:03:22');
/* Выведет: 12 */
```

- TIME() — время:

```
SELECT TIME('2018-02-12 06:03:22');
/* Выведет: 06:03:22 */
```

- HOUR() — час:

```
SELECT HOUR('2018-02-12 06:03:22');
/* Выведет: 6 */
```

- MINUTE() — минуты:

```
SELECT MINUTE('2018-02-12 06:03:22');
/* Выведет: 3 */
```

- SECOND() — секунды:

```
SELECT SECOND('2018-02-12 06:03:22');
/* Выведет: 22 */
```

- MICROSECOND() — микросекунды:

```
SELECT MICROSECOND('2018-02-12 06:03:22.123456');
/* Выведет: 123456 */
```

Вместо приведенных функций можно использовать функцию `EXTRACT()`. Формат функции:

```
EXTRACT(<Тип> FROM <Дата и время>)
```

Значения параметра `<Тип>`:

- `YEAR` — год:

```
SELECT EXTRACT(YEAR FROM '2018-02-12 06:03:22');
/* Выведет: 2018 */
```

- `YEAR_MONTH` — год и месяц:

```
SELECT EXTRACT(YEAR_MONTH FROM '2018-02-12 06:03:22');
/* Выведет: 201802 */
```

- `MONTH` — месяц:

```
SELECT EXTRACT(MONTH FROM '2018-02-12 06:03:22');
/* Выведет: 2 */
```

- `DAY` — день:

```
SELECT EXTRACT(DAY FROM '2018-02-12 06:03:22');
/* Выведет: 12 */
```

- `DAY_HOUR` — день и час:

```
SELECT EXTRACT(DAY_HOUR FROM '2018-02-12 06:03:22');
/* Выведет: 1206 */
```

- `DAY_MINUTE` — день, час и минуты:

```
SELECT EXTRACT(DAY_MINUTE FROM '2018-02-12 06:03:22');
/* Выведет: 120603 */
```

- `DAY_SECOND` — день, час, минуты и секунды:

```
SELECT EXTRACT(DAY_SECOND FROM '2018-02-12 06:03:22');
/* Выведет: 12060322 */
```

- `DAY_MICROSECOND` — день, час, минуты, секунды и микросекунды:

```
SELECT EXTRACT(DAY_MICROSECOND
FROM '2018-02-12 06:03:22.111111');
/* Выведет: 12060322111111 */
```

- `HOUR` — час:

```
SELECT EXTRACT(HOUR FROM '2018-02-12 06:03:22');
/* Выведет: 6 */
```

- `HOUR_MINUTE` — час и минуты:

```
SELECT EXTRACT(HOUR_MINUTE FROM '2018-02-12 06:03:22');
/* Выведет: 603 */
```

- `HOUR_SECOND` — час, минуты и секунды:

```
SELECT EXTRACT(HOUR_SECOND FROM '2018-02-12 06:03:22');
/* Выведет: 60322 */
```

- `HOUR_MICROSECOND` — час, минуты, секунды и микросекунды:

```
SELECT EXTRACT(HOUR_MICROSECOND
FROM '2018-02-12 06:03:22.111111');
/* Выведет: 60322111111 */
```

- `MINUTE` — минуты:

```
SELECT EXTRACT(MINUTE FROM '2018-02-12 06:03:22');
/* Выведет: 3 */
```

- `MINUTE_SECOND` — минуты и секунды:

```
SELECT EXTRACT(MINUTE_SECOND FROM '2018-02-12 06:03:22');
/* Выведет: 322 */
```

- `MINUTE_MICROSECOND` — минуты, секунды и микросекунды:

```
SELECT EXTRACT(MINUTE_MICROSECOND
FROM '2018-02-12 06:03:22.111111');
/* Выведет: 322111111 */
```

- `SECOND` — секунды:

```
SELECT EXTRACT(SECOND FROM '2018-02-12 06:03:22');
/* Выведет: 22 */
```

- `SECOND_MICROSECOND` — секунды и микросекунды:

```
SELECT EXTRACT(SECOND_MICROSECOND
FROM '2018-02-12 06:03:22.111111');
/* Выведет: 22111111 */
```

- `MICROSECOND` — микросекунды:

```
SELECT EXTRACT(MICROSECOND FROM '2018-02-12 06:03:22.111111');
/* Выведет: 111111 */
```

С помощью следующих функций можно получить дополнительные сведения о дате:

- `QUARTER()` — порядковый номер квартала в году (от `1` до `4`):

```
SELECT QUARTER('2018-02-12');
/* Выведет: 1 */
```

- `WEEK(<Дата>[, <Режим>])` — порядковый номер недели. Необязательный второй параметр задает режим вычисления номера недели: если второй параметр не задан, для него устанавливается значение `0` (при котором значение возвращается в диапазоне от `0` до `53`, неделя начинается с воскресенья):

```
SELECT WEEK('2018-02-12');
/* Выведет: 6 */
```

- `WEEKOFYEAR()` — порядковый номер недели в году (от `1` до `53`). Фактически выполняет ту же задачу, что вызов функции `WEEK()` с параметром режима `3`:

```
SELECT WEEKOFYEAR('2018-02-12');
/* Выведет: 7 */
```

- `YEARWEEK(<Дата>[, <Режим>])` — число в формате `ГГГГНН`, где `ГГГГ` — год, а `НН` — порядковый номер недели в году. Второй параметр выполняет ту же задачу, что второй параметр функции `WEEK()`:

```
SELECT YEARWEEK('2018-02-12');
/* Выведет: 201806 */
```

- `DAYOFYEAR()` — порядковый номер дня в году (от `1` до `366`):

```
SELECT DAYOFYEAR('2018-02-12');
/* Выведет: 43 */
```

- `MAKEDATE(<Год>, <Номер дня в году>)` — дата в формате `ГГГГ-ММ-ДД` по номеру дня в году:

```
SELECT MAKEDATE(2018, 43);
/* Выведет: 2018-02-12 */
```

- `DAYOFWEEK()` — порядковый номер дня недели (`1` — для воскресенья, `2` — для понедельника, ..., `7` — для субботы):

```
SELECT DAYOFWEEK('2018-02-12');
/* Выведет: 2 */
```

- `WEEKDAY()` — порядковый номер дня недели (`0` — для понедельника, `1` — для вторника, ..., `6` — для воскресенья):

```
SELECT WEEKDAY('2018-02-12');
/* Выведет: 0 */
```

- `DAYNAME()` — название дня недели в зависимости от настроек локали (по умолчанию используется английская локаль `en_US`):

```
SELECT @@lc_time_names;
/* Выведет: en_US */
SELECT DAYNAME('2018-02-12');
/* Выведет: Monday */
```

- `TO_DAYS(<Дата>)` — число дней, прошедших с нулевого года:

```
SELECT TO_DAYS('2018-02-12');
/* Выведет: 737102 */
```

- `FROM_DAYS(<Число дней>)` — дата в формате `ГГГГ-ММ-ДД` по числу дней, прошедших с нулевого года:

```
SELECT FROM_DAYS(737102);
/* Выведет: 2018-02-12 */
```

- `TIME_TO_SEC(<Время>)` — число секунд, прошедших с начала суток:

```
SELECT TIME_TO_SEC('16:15:24');
/* Выведет: 58524 */
```

- `SEC_TO_TIME(<Число секунд>)` — время в формате `ЧЧ:ММ:СС` по числу секунд, прошедших с начала суток:

```
SELECT SEC_TO_TIME(58524);
/* Выведет: 16:15:24 */
```

Для манипуляции датой и временем можно использовать следующие функции:

- ADDDATE(<Дата>, INTERVAL <Интервал> <Тип>) и DATE_ADD(<Дата>, INTERVAL <Интервал> <Тип>) — прибавляют к параметру <Дата> временной интервал;
- SUBDATE(<Дата>, INTERVAL <Интервал> <Тип>) и DATE_SUB(<Дата>, INTERVAL <Интервал> <Тип>) — вычитают из параметра <Дата> временной интервал.

  Параметр <Тип> в функциях ADDDATE(), DATE_ADD(), SUBDATE() и DATE_SUB() может принимать следующие значения:

  - YEAR — год:

    ```
    SELECT ADDDATE('2018-02-12 06:03:22', INTERVAL 2 YEAR);
    /* Выведет: 2020-02-12 06:03:22 */
    ```

  - YEAR_MONTH — год и месяц (формат 'ГГ-ММ'):

    ```
    SELECT ADDDATE('2018-02-12 06:03:22', INTERVAL '2-2' YEAR_MONTH);
    /* Выведет: 2020-04-12 06:03:22 */
    ```

  - MONTH — месяц:

    ```
    SELECT ADDDATE('2018-02-12 06:03:22', INTERVAL 3 MONTH);
    /* Выведет: 2018-05-12 06:03:22 */
    ```

  - QUARTER — квартал:

    ```
    SELECT ADDDATE('2018-02-12 06:03:22', INTERVAL 1 QUARTER);
    /* Выведет: 2018-05-12 06:03:22 */
    ```

  - WEEK — неделя:

    ```
    SELECT ADDDATE('2018-02-12 06:03:22', INTERVAL 3 WEEK);
    /* Выведет: 2018-03-05 06:03:22 */
    ```

  - DAY — день:

    ```
    SELECT ADDDATE('2018-02-12 06:03:22', INTERVAL 6 DAY);
    /* Выведет: 2018-02-18 06:03:22 */
    ```

  - DAY_HOUR — день и час (формат 'ДД ЧЧ'):

    ```
    SELECT ADDDATE('2018-02-12 06:03:22',
    INTERVAL '6 3' DAY_HOUR);
    /* Выведет: 2018-02-18 09:03:22 */
    ```

  - DAY_MINUTE — день, час и минуты (формат 'ДД ЧЧ:ММ'):

    ```
    SELECT ADDDATE('2018-02-12 06:03:22',
    INTERVAL '6 3:5' DAY_MINUTE);
    /* Выведет: 2018-02-18 09:08:22 */
    ```

  - DAY_SECOND — день, час, минуты и секунды (формат 'ДД ЧЧ:ММ:СС'):

    ```
    SELECT ADDDATE('2018-02-12 06:03:22',
    INTERVAL '6 3:5:15' DAY_SECOND);
    /* Выведет: 2018-02-18 09:08:37 */
    ```

- `DAY_MICROSECOND` — день, час, минуты, секунды и микросекунды (формат `'ДД ЧЧ:ММ:СС.XXXXXX'`):

```
SELECT ADDDATE('2018-02-12 06:03:22',
INTERVAL '6 3:5:15.10' DAY_MICROSECOND);
/* Выведет: 2018-02-18 09:08:37.100000 */
```

- `HOUR` — час:

```
SELECT ADDDATE('2018-02-12 06:03:22', INTERVAL 3 HOUR);
/* Выведет: 2018-02-12 09:03:22 */
```

- `HOUR_MINUTE` — час и минуты (формат `'ЧЧ:ММ'`):

```
SELECT ADDDATE('2018-02-12 06:03:22',
INTERVAL '3:7' HOUR_MINUTE);
/* Выведет: 2018-02-12 09:10:22 */
```

- `HOUR_SECOND` — час, минуты и секунды (формат `'ЧЧ:ММ:СС'`):

```
SELECT ADDDATE('2018-02-12 06:03:22',
INTERVAL '3:7:15' HOUR_SECOND);
/* Выведет: 2018-02-12 09:10:37 */
```

- `HOUR_MICROSECOND` — час, минуты, секунды и микросекунды (формат `'ЧЧ:ММ:СС.XXXXXX'`):

```
SELECT ADDDATE('2018-02-12 06:03:22',
INTERVAL '3:7:15.10' HOUR_MICROSECOND);
/* Выведет: 2018-02-12 09:10:37.100000 */
```

- `MINUTE` — минуты:

```
SELECT ADDDATE('2018-02-12 06:03:22', INTERVAL 8 MINUTE);
/* Выведет: 2018-02-12 06:11:22 */
```

- `MINUTE_SECOND` — минуты и секунды (формат `'ММ:СС'`):

```
SELECT ADDDATE('2018-02-12 06:03:22',
INTERVAL '3:7' MINUTE_SECOND);
/* Выведет: 2018-02-12 06:06:29 */
```

- `MINUTE_MICROSECOND` — минуты, секунды и микросекунды (формат `'ММ:СС.XXXXXX'`):

```
SELECT ADDDATE('2018-02-12 06:03:22',
INTERVAL '3:7.11' MINUTE_MICROSECOND);
/* Выведет: 2018-02-12 06:06:29.110000 */
```

- `SECOND` — секунды:

```
SELECT ADDDATE('2018-02-12 06:03:22', INTERVAL 15 SECOND);
/* Выведет: 2018-02-12 06:03:37 */
```

- `SECOND_MICROSECOND` — секунды и микросекунды (формат `'СС.XXXXXX'`):

```
SELECT ADDDATE('2018-02-12 06:03:22',
INTERVAL '15.123456' SECOND_MICROSECOND);
/* Выведет: 2018-02-12 06:03:37.123456 */
```

- `MICROSECOND` — микросекунды:

  ```
  SELECT ADDDATE('2018-02-12 06:03:22',
  INTERVAL 123456 MICROSECOND);
  /* Выведет: 2018-02-12 06:03:22.123456 */
  ```

- `ADDDATE(<Дата>, <Интервал в днях>)` — прибавляет к параметру `<Дата>` временной интервал в днях. Если указать перед интервалом знак `-`, то интервал вычитается из даты:

  ```
  SELECT ADDDATE('2018-02-12', 10);
  /* Выведет: 2018-02-22 */
  SELECT ADDDATE('2018-02-12', -10);
  /* Выведет: 2018-02-02 */
  ```

- `SUBDATE(<Дата>, <Интервал в днях>)` — вычитает из параметра `<Дата>` временной интервал в днях. Если указать перед интервалом знак `-`, то интервал прибавляется к дате:

  ```
  SELECT SUBDATE('2018-02-12', 10);
  /* Выведет: 2018-02-02 */
  SELECT SUBDATE('2018-02-12', -10);
  /* Выведет: 2018-02-22 */
  ```

- `ADDTIME(<Дата>, <Время>)` — прибавляет к параметру `<Дата>` временной интервал:

  ```
  SELECT ADDTIME('2018-02-12 06:03:22', '12:52:35');
  /* Выведет: 2018-02-12 18:55:57 */
  ```

- `SUBTIME(<Дата>, <Время>)` — вычитает из параметра `<Дата>` временной интервал:

  ```
  SELECT SUBTIME('2018-02-12 06:03:22', '12:52:35');
  /* Выведет: 2018-02-11 17:10:47 */
  ```

- `DATEDIFF(<Конечная дата>, <Начальная дата>)` — вычисляет число дней между двумя датами:

  ```
  SELECT DATEDIFF('2018-02-12', '2018-01-03');
  /* Выведет: 40 */
  ```

- `TIMEDIFF(<Конечная дата>, <Начальная дата>)` — вычисляет разницу между двумя временными значениями:

  ```
  SELECT TIMEDIFF('16:15:24', '08:43:17');
  /* Выведет: 07:32:07 */
  SELECT TIMEDIFF('2018-02-12 22:36:43', '2018-02-12 15:36:43');
  /* Выведет: 07:00:00 */
  ```

- `PERIOD_ADD(<Дата>, <Число месяцев>)` — добавляет заданное `<Число месяцев>` к дате, заданной в формате `ГГГГММ` или `ГГММ`:

  ```
  SELECT PERIOD_ADD(201802, 4);
  /* Выведет: 201806 */
  ```

- `PERIOD_DIFF(<Конечная дата>, <Начальная дата>)` — вычисляет разницу в месяцах между двумя временными значениями, заданными в формате `ГГГГММ` или `ГГММ`:

```
SELECT PERIOD_DIFF(201812, 201810);
/* Выведет: 2 */
```

- `CONVERT_TZ(<Дата>, <Часовой пояс 1>, <Часовой пояс 2>)` — переводит дату из одного часового пояса в другой:

```
SELECT CONVERT_TZ('2018-02-12 06:03:22', '+00:00', '+4:00');
/* Выведет: 2018-02-12 10:03:22 */
```

- `LAST_DAY(<Дата>)` — возвращает дату в формате `ГГГГ-ММ-ДД`, в которой день выставлен на последний день текущего месяца:

```
SELECT LAST_DAY('2018-02-12 06:03:22');
/* Выведет: 2018-02-28 */
```

- `MAKETIME(<Часы>, <Минуты>, <Секунды>)` — возвращает время в формате `ЧЧ:ММ:СС`:

```
SELECT MAKETIME(12, 52, 35);
/* Выведет: 12:52:35 */
```

- `TIMESTAMP(<Дата>[, <Время>])` — возвращает дату в формате `ГГГГ-ММ-ДД ЧЧ:ММ:СС`:

```
SELECT TIMESTAMP('2018-02-12');
/* Выведет: 2018-02-12 00:00:00 */
SELECT TIMESTAMP('2018-02-12', '13:48:11');
/* Выведет: 2018-02-12 13:48:11 */
```

Помимо описанных функций, добавить или вычесть интервал времени можно с помощью операторов `+` и `-`, за которыми следует ключевое слово `INTERVAL`, значение и тип интервала. Применимы те же типы интервалов, что и в функциях `ADDDATE()`, `DATE_ADD()`, `SUBDATE()` и `DATE_SUB()`:

```
SELECT '2018-02-12 06:03:22' + INTERVAL '3:7:15' HOUR_SECOND;
/* Выведет: 2018-02-12 09:10:37 */
SELECT '2018-02-12 06:03:22' - INTERVAL '3:7:15' HOUR_SECOND;
/* Выведет: 2018-02-12 02:56:07 */
```

Для форматирования даты и времени предназначено несколько функций:

- `DATE_FORMAT(<Дата>, <Формат>)` — форматирует дату в соответствии со строкой `<Формат>`:

```
SELECT DATE_FORMAT('2018-02-12 06:03:22', '%d.%m.%Y');
/* Выведет: 12.02.2018 */
```

- `STR_TO_DATE(<Строка>, <Формат>)` — возвращает дату и (или) время, соответствующую строке `<Формат>`:

```
SELECT STR_TO_DATE('20.11.2018 13:48:11', '%d.%m.%Y %H:%i:%s');
/* Выведет: 2018-11-20 13:48:11 */
```

```
SELECT STR_TO_DATE('20.11.2018', '%d.%m.%Y');
/* Выведет: 2018-11-20 */
SELECT STR_TO_DATE('13:48:11', '%H:%i:%s');
/* Выведет: 13:48:11 */
```

- `TIME_FORMAT(<Время>, <Формат>)` — форматирует время в соответствии со строкой `<Формат>`:

```
SELECT TIME_FORMAT('16:15:24', '%H %i %s');
/* Выведет: 16 15 24 */
```

- `FROM_UNIXTIME(<Дата>[, <Формат>])` — возвращает дату в формате `ГГГГ-ММ-ДД ЧЧ:ММ:СС` или соответствующую строке `<Формат>` по числу секунд, прошедших с полуночи 1 января 1970 г.:

```
SELECT FROM_UNIXTIME(1518451863);
/* Выведет: 2018-02-12 19:11:03 */
SELECT FROM_UNIXTIME(1518451863, '%d.%m.%Y');
/* Выведет: 12.02.2018 */
```

- `GET_FORMAT(<Тип времени>, '<Стандарт>')` — возвращает строку форматирования для пяти стандартов отображения даты и времени. Параметр `<Тип времени>` может принимать следующие значения:
  - `DATETIME` — дата и время;
  - `DATE` — дата;
  - `TIME` — время.

  Параметр `<Стандарт>` может принимать такие значения:
  - `ISO` — стандарт ISO;
  - `EUR` — европейский стандарт;
  - `USA` — американский стандарт;
  - `JIS` — японский стандарт;
  - `INTERNAL` — внутренний формат MySQL.

  Пример:

```
SELECT GET_FORMAT(DATE, 'EUR');
/* Выведет: %d.%m.%Y */
SELECT DATE_FORMAT('2018-02-12 06:03:22',
GET_FORMAT(DATE, 'EUR'));
/* Выведет: 12.02.2018 */
```

Параметр `<Формат>` в функциях форматирования может содержать следующие комбинации символов:

- `%Y` — год из четырех цифр;
- `%y` — год из двух цифр;
- `%m` — номер месяца с предваряющим нулем (от `01` до `12`);

- `%c` — номер месяца без предваряющего нуля (от `1` до `12`);
- `%b` — аббревиатура месяца из трех букв (зависит от локали);
- `%M` — полное название месяца (зависит от локали);
- `%d` — номер дня с предваряющим нулем (от `01` до `31`);
- `%e` — номер дня без предваряющего нуля (от `1` до `31`);
- `%w` — номер дня недели (`0` — для воскресенья и `6` — для субботы);
- `%a` — аббревиатура дня недели из трех букв (зависит от локали);
- `%W` — полное название дня недели (зависит от локали);
- `%H` — часы в 24-часовом формате (от `00` до `23`);
- `%k` — часы в 24-часовом формате (от `0` до `23`);
- `%h` и `%I` — часы в 12-часовом формате (от `01` до `12`);
- `%l` — часы в 12-часовом формате (от `1` до `12`);
- `%i` — минуты (от `00` до `59`);
- `%s` — секунды (от `00` до `59`);
- `%f` — микросекунды;
- `%%` — знак процента.

Настройки параметра `<Формат>` зависят от значения переменной `lc_time_names`. Выведем текущее значение переменной:

```
SELECT @@lc_time_names;
/* Выведет: en_US */
```

В качестве примера изменим значение переменной и выведем название месяца на русском языке:

```
SELECT DATE_FORMAT('2018-02-12 06:03:22', '%d %M %Y');
/* Выведет: 12 February 2018 */
SET lc_time_names = 'ru_RU';
SELECT DATE_FORMAT('2018-02-12 06:03:22', '%d %M %Y');
/* Выведет: 12 Февраля 2018 */
```

6.11.3. Функции для обработки строк

Приведем основные функции для обработки строк:

- `CHAR_LENGTH(<Строка>)` и `CHARACTER_LENGTH(<Строка>)` — возвращают длину строки в символах. Функции корректно работают с многобайтовыми кодировками:

  ```
  SELECT CHAR_LENGTH('Строка');
  /* Выведет: 6 */
  SELECT CHARACTER_LENGTH('Строка');
  /* Выведет: 6 */
  ```

- `LENGTH(<Строка>)` — возвращает длину строки в байтах (вследствие чего длина строк, записанных в многобайтовых кодировках, определяется некорректно):

```
SELECT LENGTH('Строка');
/* Выведет (cp1251): 6 */
/* Выведет (utf8): 12 */
```

- `BIT_LENGTH(<Строка>)` — возвращает длину строки в битах:

```
SELECT BIT_LENGTH('Строка');
/* Выведет (cp1251): 48 */
/* Выведет (utf8): 96 */
```

- `CONCAT(<Строка 1>, <Строка 2>, ..., <Строка N>)` — объединяет все параметры в одну строку:

```
SELECT CONCAT('string1', 'string2', 'string3');
/* Выведет: string1string2string3 */
```

- `CONCAT_WS(<Разделитель>, <Строка 1>, ..., <Строка N>)` — объединяет все параметры в одну строку через разделитель, заданный в параметре `<Разделитель>`:

```
SELECT CONCAT_WS(' - ', 'string1', 'string2', 'string3');
/* Выведет: string1 - string2 - string3 */
```

- `TRIM([[<Откуда>] [<Символы для удаления>] FROM] <Строка>)` — удаляет из начала (и/или конца) строки символы, указанные в параметре `<Символы для удаления>`. Если параметр не указан, то удаляются пробелы. Необязательный параметр `<Откуда>` может принимать значения:
  - `BOTH` — символы удаляются из начала и конца строки (по умолчанию);
  - `LEADING` — только из начала строки;
  - `TRAILING` — только из конца строки.

  Примеры:

```
SELECT CONCAT("'", TRIM('   String   '), "'");
/* Выведет: 'String' */
SELECT CONCAT("'", TRIM(LEADING FROM '   String   '), "'");
/* Выведет: 'String   ' */
SELECT CONCAT("'", TRIM(TRAILING FROM '   String   '), "'");
/* Выведет: '   String' */
SELECT CONCAT("'", TRIM(BOTH 'm' FROM 'mmmmStringmmmm'), "'");
/* Выведет: 'String' */
SELECT CONCAT("'", TRIM(TRAILING 'ing' FROM 'Stringing'), "'");
/* Выведет: 'Str' */
SELECT CONCAT("'", TRIM(TRAILING 'gn' FROM 'String'), "'");
/* Выведет: 'String' */
```

- `LTRIM(<Строка>)` — удаляет пробелы в начале строки:

```
SELECT CONCAT("'", LTRIM('   String   '), "'");
/* Выведет: 'String   ' */
```

- `RTRIM(<Строка>)` — удаляет пробелы в конце строки:

```
SELECT CONCAT("'", RTRIM('   String   '), "'");
/* Выведет: '   String' */
```

- `LOWER(<Строка>)` и `LCASE(<Строка>)` — переводят все символы в нижний регистр:

```
SELECT LOWER('СТРОКА');
/* Выведет: строка */
SELECT LCASE('СТРОКА');
/* Выведет: строка */
```

- `UPPER(<Строка>)` и `UCASE(<Строка>)` — переводят все символы в верхний регистр:

```
SELECT UPPER('строка');
/* Выведет: СТРОКА */
SELECT UCASE('строка');
/* Выведет: СТРОКА */
```

- `REVERSE(<Строка>)` — возвращает строку в обратном порядке:

```
SELECT REVERSE('string');
/* Выведет: gnirts */
```

- `LEFT(<Строка>, <Число символов>)` — возвращает заданное число крайних символов слева:

```
SELECT LEFT('string', 2);
/* Выведет: st */
```

- `RIGHT(<Строка>, <Число символов>)` — возвращает заданное число крайних символов справа:

```
SELECT RIGHT('string', 2);
/* Выведет: ng */
```

- `SUBSTRING(<Строка>, <Начальная позиция>[, <Длина>])`, `SUBSTR(<Строка>, <Начальная позиция>[, <Длина>])` и `MID(<Строка>, <Начальная позиция>[, <Длина>])` — позволяют получить подстроку заданной длины, начиная с позиции `<Начальная позиция>`. Если параметр `<Длина>` не задан, то возвращаются все символы до конца строки:

```
SELECT SUBSTRING('string', 2, 2);
/* Выведет: tr */
SELECT SUBSTR('string', 2, 2);
/* Выведет: tr */
SELECT MID('string', 2);
/* Выведет: tring */
```

Первые две функции имеют альтернативный синтаксис:

```
SELECT SUBSTRING('string' FROM 2 FOR 3);
/* Выведет: tri */
SELECT SUBSTRING('string' FROM 2);
/* Выведет: tring */
```

- `LPAD(<Строка>, <Длина>, <Подстрока>)` — добавляет подстроку к исходной строке слева, доводя общую длину строки до величины `<Длина>`:

```
SELECT LPAD('string', 11, 'mp');
/* Выведет: mpmpmstring */
```

- `RPAD(<Строка>, <Длина>, <Подстрока>)` — добавляет подстроку к исходной строке справа, доводя общую длину строки до величины `<Длина>`:

```
SELECT RPAD('string', 10, 'mp');
/* Выведет: stringmpmp */
```

- `REPEAT(<Строка>, <Число повторений>)` — возвращает строку, содержащую заданное число повторений исходной строки:

```
SELECT REPEAT('str', 3);
/* Выведет: strstrstr */
```

- `SPACE(<Число пробелов>)` — возвращает строку, состоящую из заданного числа пробелов:

```
SELECT CONCAT("'", SPACE(3), 'String', "'");
/* Выведет: '   String' */
```

- `ELT(<Номер из списка>, <Строка 1>, ..., <Строка N>)` — позволяет получить одну строку из списка параметров, номер которой задается первым параметром:

```
SELECT ELT(2, 'string1', 'string2', 'string3');
/* Выведет: string2 */
```

- `ASCII(<Строка>)` — возвращает код ASCII первого символа строки:

```
SELECT ASCII('String');
/* Выведет: 83 */
```

- `ORD(<Строка>)` — дает возможность узнать код первого символа строки. Корректно работает с многобайтовыми кодировками. Если первый символ — однобайтовый, вернет то же значение, что и `ASCII()`:

```
SELECT ORD('String');
/* Выведет: 83 */
```

- `CHAR(<ASCII-код 1>, <ASCII-код 2>, ..., <ASCII-код N> [USING <Кодировка>])` — возвращает строку, состоящую из последовательности символов, соответствующих ASCII-кодам:

```
SELECT CHAR(83, 116, 114, 105, 110, 103);
/* Выведет: String */
```

- `INSTR(<Строка>, <Подстрока>)` или `POSITION(<Подстрока> IN <Строка>)` — ищут подстроку в строке и возвращают позицию ее первого вхождения. Если вхождение не найдено, то возвращается 0:

```
SELECT INSTR('string', 'st');
/* Выведет: 1 */
```

```
SELECT POSITION('st' IN 'string');
/* Выведет: 1 */
SELECT POSITION('pt' IN 'string');
/* Выведет: 0 */
```

- LOCATE(<Подстрока>, <Строка>[, <Начальная позиция>]) — возвращает позицию первого вхождения подстроки в строку, начиная с указанной начальной позиции. Если подстрока не найдена, то возвращается 0. Если начальная позиция не указана, то поиск производится с начала строки:

```
SELECT LOCATE('st', 'string_st');
/* Выведет: 1 */
SELECT LOCATE('st', 'string_st', 3);
/* Выведет: 8 */
```

- FIELD(<Исходная строка>, <Строка 1>, ..., <Строка N>) — позволяет определить номер строки из списка <Строка 1>, ..., <Строка N>, которая совпадает с исходной строкой:

```
SELECT FIELD('st', 'string', 'st', 'st2');
/* Выведет: 2 */
```

- FIND_IN_SET(<Исходная строка>, <Список подстрок через запятую>) — возвращает номер строки из списка <Список подстрок через запятую>, которая совпадает с исходной строкой:

```
SELECT FIND_IN_SET('st', 'string,st,st2');
/* Выведет: 2 */
```

- REPLACE(<Строка>, <Подстрока для замены>, <Новая подстрока>) — заменяет все вхождения подстроки на новую подстроку и возвращает результат:

```
SELECT REPLACE('Привет, Петя', 'Петя', 'Вася');
/* Выведет: Привет, Вася */
```

- SUBSTRING_INDEX(<Строка>, <Подстрока>, <Номер вхождения>) — находит N-е вхождение подстроки в строку, где N задается параметром <Номер вхождения>, и возвращает часть строки, расположенную слева от подстроки:

```
SELECT SUBSTRING_INDEX('синий, красный, зеленый', ',', 1);
/* Выведет: синий */
SELECT SUBSTRING_INDEX('синий, красный, зеленый', ',', 2);
/* Выведет: синий, красный */
```

  Если параметр <Номер вхождения> имеет отрицательное значение, то ищется N-е вхождение подстроки с конца строки и возвращается часть строки, расположенная справа от найденной подстроки:

```
SELECT CONCAT('"',
SUBSTRING_INDEX('синий, красный, зеленый', ',', -1), '"');
/* Выведет: " зеленый" */
SELECT CONCAT('"',
SUBSTRING_INDEX('синий, красный, зеленый', ',', -2), '"');
/* Выведет: " красный, зеленый" */
```

- ❒ `INSERT(<Строка>, <Начальная позиция>, <Длина>, <Подстрока>)` — заменяет фрагмент в строке с начальной позиции длиной `<Длина>` на значение параметра `<Подстрока>`:

```
SELECT INSERT('красный', 6, 2, 'ое');
/* Выведет: красное */
SELECT INSERT('красный', 6, 1, 'ое');
/* Выведет: красноей */
```

- ❒ `QUOTE(<Строка>)` — экранирует все специальные символы в строке:

```
SELECT QUOTE("Д'Артаньян и три мушкетера");
/* Выведет: Д\'Артаньян и три мушкетера */
```

- ❒ `UNHEX(<Строка>)` — переводит строку из шестнадцатеричных цифр в обычную строку. Каждая пара символов в исходной строке воспринимается как шестнадцатеричное число, которое преобразуется в символ:

```
SELECT UNHEX('537472696E67');
/* Выведет: String */
```

- ❒ `COMPRESS(<Строка>)` — архивирует строку. Сжатую строку следует хранить в полях, имеющих бинарный тип данных;
- ❒ `UNCOMPRESS(<Строка>)` — разархивирует строку, сжатую функцией `COMPRESS()`:

```
SELECT UNCOMPRESS(COMPRESS('Строка'));
/* Выведет: Строка */
```

- ❒ `UNCOMPRESSED_LENGTH(<Строка>)` — позволяет узнать длину строки в байтах, которую она будет иметь после разархивирования:

```
SELECT UNCOMPRESSED_LENGTH(COMPRESS('Строка'));
/* Выведет (cp1251): 6 */
/* Выведет (utf8): 12 */
```

- ❒ `CHARSET(<Строка>)` — возвращает название кодировки для строки:

```
SELECT CHARSET('Строка');
/* Выведет (в консоли): cp1251 */
/* Выведет (в phpMyAdmin): utf8mb4 */
```

- ❒ `COLLATION(<Строка>)` — возвращает порядок сравнения для строки:

```
SELECT COLLATION('Строка');
/* Выведет (в консоли): cp1251_general_ci */
/* Выведет (в phpMyAdmin): utf8mb4_unicode_ci */
```

- ❒ `STRCMP(<Строка 1>, <Строка 2>)` — сравнивает две строки и возвращает:
  - `0` — если строки идентичны;
  - `-1` — если `<Строка 1>` больше `<Строка 2>`;
  - `1` — если `<Строка 1>` меньше `<Строка 2>`.

Примеры:

```
SELECT STRCMP('Строка', 'Строка');
/* Выведет: 0 */
SELECT STRCMP('Строка1', 'Строка2');
/* Выведет: -1 */
SELECT STRCMP('Строка2', 'Строка1');
/* Выведет: 1 */
```

Сравнение строк чувствительно к регистру;

- `LOAD_FILE(<Путь к файлу>)` — возвращает содержимое файла в виде строки. Часто используется для заполнения бинарных полей. В качестве примера создадим текстовый файл с названием test.txt в каталоге C:\xampp\htdocs. Затем запишем в файл строку `Content`. Теперь получим содержимое файла с помощью функции `LOAD_FILE()`:

```
SELECT LOAD_FILE('C:/xampp/htdocs/test.txt');
/* Выведет: Content */
```

6.11.4. Функции для шифрования строк

Для необратимого шифрования применяются следующие функции:

- `MD5(<Строка>)` — кодирует строку по алгоритму `MD5`. Возвращает шестнадцатеричное число, содержащее 32 шестнадцатеричные цифры:

```
SELECT MD5('password');
/* Выведет: 5f4dcc3b5aa765d61d8327deb882cf99 */
```

- `SHA(<Строка>)` и `SHA1(<Строка>)` — возвращают 40-разрядное шестнадцатеричное число:

```
SELECT SHA('password');
/* Выведет: 5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8 */
SELECT SHA1('password');
/* Выведет: 5baa61e4c9b93f3f0682250b6cf8331b7ee68fd8 */
```

- `SHA2(<Строка>, <Длина бита результата>)` — возвращает шестнадцатеричное число. Во втором параметре можно указать значения `224`, `256`, `384`, `512` или `0` (то же самое, что `256`):

```
SELECT SHA2('password', 0);
/* Выведет:
5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d1542d8
*/
```

Для симметричного шифрования применяются следующие функции:

- `AES_ENCRYPT(<Строка>, <Ключ>)` — принимает строку и секретный ключ и возвращает бинарную строку, зашифрованную по алгоритму `AES`;
- `AES_DECRYPT(<Зашифрованная строка>, <Ключ>)` — служит для расшифровки строк, зашифрованных функцией `AES_ENCRYPT()`:

```
SET @my_key = SHA2('Мой ключ', 512);
SELECT AES_DECRYPT(AES_ENCRYPT('password', @my_key), @my_key);
/* Выведет: password */
```

6.11.5. Информационные функции

Приведем перечень информационных функций:

- `VERSION()` — выводит информацию о версии сервера:

```
SELECT VERSION();
/* Выведет: 10.1.29-MariaDB */
```

- `USER()` — позволяет узнать имя пользователя и имя хоста текущего пользователя:

```
SELECT USER();
/* Выведет: root@localhost */
```

- `CURRENT_USER()` — выдает имя пользователя и имя хоста текущего пользователя в сессии:

```
SELECT CURRENT_USER();
/* Выведет: root@localhost */
```

- `DATABASE()` — возвращает название текущей базы данных:

```
SELECT DATABASE();
/* Выведет: tests */
```

- `CONNECTION_ID()` — возвращает идентификатор соединения:

```
SELECT CONNECTION_ID();
/* Выведет: 12 */
```

- `DEFAULT(<Имя поля>)` — позволяет узнать значение по умолчанию для указанного поля:

```
CREATE TABLE `new_table` (
`id` INT AUTO_INCREMENT,
`count` INT DEFAULT 25,
PRIMARY KEY(`id`)
) ENGINE=MyISAM CHARSET=utf8 COLLATE utf8_general_ci;
INSERT INTO `new_table` VALUES(NULL, 50);
SELECT DEFAULT(`count`) FROM `new_table` LIMIT 1;
/* Выведет: 25 */
```

- `LAST_INSERT_ID()` — служит для определения последнего автоматически сгенерированного значения для поля с атрибутом `AUTO_INCREMENT`. Значение возвращается только в том случае, если перед вызовом функции было сгенерировано новое значение:

```
INSERT INTO `new_table` VALUES(NULL, 80);
SELECT LAST_INSERT_ID();
/* Выведет: 2 */
```

Вывести последнюю добавленную запись можно так:

```
INSERT INTO `new_table` VALUES(NULL, 30);
SELECT `count` FROM `new_table` WHERE `id` = LAST_INSERT_ID();
/* Выведет: 30 */
```

- `FOUND_ROWS()` — позволяет узнать число строк, которое возвратил бы оператор `SELECT` без инструкции `LIMIT`.

  Чтобы получить значение, необходимо в операторе `SELECT` указать опцию `SQL_CALC_FOUND_ROWS`:

```
INSERT INTO `new_table` VALUES(NULL, 6), (NULL, 70), (NULL, 50);
SELECT COUNT(*) FROM `new_table`;
/* Выведет: 6 */
SELECT SQL_CALC_FOUND_ROWS `count` FROM `new_table` LIMIT 0, 3;
/* Выведет три записи: 50, 80, 30 */
SELECT FOUND_ROWS();
/* Выведет: 6 */
```

- `BENCHMARK(<Число повторений>, <SQL-запрос>)` — выполняет SQL-запрос заданное число раз. Функция всегда возвращает значение `0`. Применяется для определения быстродействия SQL-запроса:

```
SELECT BENCHMARK(1000000, MD5('строка'));
/* Выведет: 0
1 row in set (0.26 sec) */
```

6.11.6. Прочие функции

Также в SQL-запросах можно использовать следующие функции:

- `IF(<Условие>, <Если Истина>, <Если Ложь>)` — функция для логического выбора. Если `<Условие>` истинно, то возвращается значение выражения `<Если Истина>`, в противном случае возвращается значение выражения `<Если Ложь>`:

```
SELECT IF(5>6, 'Больше', 'Меньше');
/* Выведет: Меньше */
```

- `CASE` — оператор для логического выбора. Имеет две формы записи:
  - первая форма:

```
CASE <Переменная или выражение>
WHEN <Значение 1> THEN <Выражение 1>
[WHEN <Значение 2> THEN <Выражение 2>]
...
[ELSE <Выражение>] END
```

    В зависимости от значения переменной (или выражения) выполняется один из блоков `WHEN`, в котором указано это значение. Если ни одно из значений не описано в блоках `WHEN`, то выполняется блок `ELSE`:

```
SELECT CASE 3 + 5 WHEN 8 THEN 'Равно 8'
WHEN 7 THEN 'Равно 7' ELSE 'Не смогли определить' END;
/* Выведет: Равно 8 */
```

- вторая форма:

```
CASE WHEN <Условие 1> THEN <Выражение 1>
[WHEN <Условие 2> THEN <Выражение 2>]
...
[ELSE <Выражение>] END
```

Пример:

```
SELECT CASE WHEN 5>6 THEN 'Больше' ELSE 'Меньше' END;
/* Выведет: Меньше */
```

- `IFNULL(<Выражение1>, <Выражение2>)` — позволяет заменить значения `NULL` другими значениями. Если `<Выражение1>` не равно `NULL`, то функция возвращает `<Выражение1>`. В противном случае функция возвращает `<Выражение2>`:

```
SELECT IFNULL(5, 3);
/* Выведет: 5 */
SELECT IFNULL(NULL, 3);
/* Выведет: 3 */
```

- `NULLIF(<Выражение1>, <Выражение2>)` — функция для логического выбора. Если `<Выражение1>` равно `<Выражение2>`, возвращается значение `NULL`, в противном случае возвращается `<Выражение1>`:

```
SELECT NULLIF(5, 5);
/* Выведет: NULL */
SELECT NULLIF(5, 3);
/* Выведет: 5 */
```

- `INET_ATON(<IP-адрес>)` — представляет IP-адрес в виде целого числа:

```
SELECT INET_ATON('127.0.0.1');
/* Выведет: 2130706433 */
```

- `INET_NTOA(<IP-адрес в виде числа>)` — принимает IP-адрес в виде целого числа и возвращает IP-адрес в виде строки, состоящей из четырех цифр, разделенных точкой:

```
SELECT INET_NTOA(2130706433);
/* Выведет: 127.0.0.1 */
```

- `GET_LOCK(<Имя>, <Время ожидания ответа сервера>)` — устанавливает блокировку с указанным именем. Функция возвращает `1` в случае успешной блокировки и `0`, если время ожидания ответа сервера превысило величину, заданную в секундах параметром `<Время ожидания ответа сервера>`. Если произошла ошибка, то функция возвращает `NULL`. Блокировка снимается тремя способами:
  - с помощью функции `RELEASE_LOCK()`;
  - при повторном вызове функции `GET_LOCK()` (до версии 5.7.5);
  - при разрыве соединения с сервером.

Например:

```
SELECT GET_LOCK('mylock', 5);
/* Выведет: 1 */
```

- `IS_FREE_LOCK(<Имя блокировки>)` — проверяет, свободна ли блокировка с указанным именем. Функция возвращает `1`, если блокировка свободна, и `0`, если она занята:

```
SELECT IS_FREE_LOCK('mylock');
/* Выведет: 0 */
```

- `IS_USED_LOCK(<Имя блокировки>)` — проверяет, установлена ли блокировка с указанным именем. Если блокировка установлена, то возвращается идентификатор соединения клиента, который установил блокировку:

```
SELECT IS_USED_LOCK('mylock');
/* Выведет: 12 */
SELECT CONNECTION_ID();
/* Выведет: 12 */
```

  Если блокировка не установлена, то возвращается значение `NULL`;

- `RELEASE_LOCK(<Имя блокировки>)` — снимает блокировку с указанным именем. Если блокировка успешно снята, то функция возвращает `1`. Если блокировка не может быть снята, то возвращается `0`. Если блокировка с указанным именем не существует, то функция возвращает `NULL`:

```
SELECT RELEASE_LOCK('mylock');
/* Выведет: 1 */
SELECT IS_USED_LOCK('mylock');
/* Выведет: NULL */
```

- `UUID()` — возвращает универсальный уникальный идентификатор — 128-разрядное уникальное число в виде строки, состоящее из пяти шестнадцатеричных чисел, разделенных символом `-`:

```
SELECT UUID();
/* Выведет: 86871720-c388-11e7-a8e7-d850e60c4cd2 */
SELECT UUID();
/* Выведет: 8687a54b-c388-11e7-a8e7-d850e60c4cd2 */
```

  Используемый алгоритм гарантирует глобальную уникальность возвращенного идентификатора;

- `UUID_SHORT()` — возвращает сокращенный уникальный идентификатор — 64-разрядное уникальное число в виде строки:

```
SELECT UUID_SHORT();
/* Выведет: 97391527101202432 */
```

- `GROUP_CONCAT()` — объединяет отдельные значения в одну строку. Формат функции:

```
GROUP_CONCAT([DISTINCT] <Поле1> [, <ПолеN>]
[ORDER BY <Поле> [ASC | DESC]]
[SEPARATOR <Разделитель>]
)
```

Для примера создадим таблицу `concat_table` в базе данных `tests`:

```
CREATE TABLE `concat_table` (
   `id` INT NOT NULL AUTO_INCREMENT,
   `counter` INT,
   PRIMARY KEY (`id`)
) ENGINE=MyISAM  CHARSET=utf8 COLLATE utf8_general_ci;
```

Затем добавим несколько записей:

```
INSERT INTO `concat_table` VALUES (NULL, 10);
INSERT INTO `concat_table` VALUES (NULL, 20);
INSERT INTO `concat_table` VALUES (NULL, 30);
INSERT INTO `concat_table` VALUES (NULL, 40);
INSERT INTO `concat_table` VALUES (NULL, 20);
```

Теперь продемонстрируем возможности функции `GROUP_CONCAT()`. Выведем все значения поля `counter`:

```
SELECT GROUP_CONCAT(`counter`) FROM `concat_table`;
/* Выведет: 10,20,30,40,20 */
```

А теперь выведем только уникальные значения, отсортированные в порядке убывания:

```
SELECT GROUP_CONCAT(DISTINCT `counter` ORDER BY `counter` DESC)
FROM `concat_table`;
/* Выведет: 40,30,20,10 */
```

И, наконец, выведем все значения поля `counter` больше `10` через разделитель `'; '`:

```
SELECT GROUP_CONCAT(DISTINCT `counter`
ORDER BY `counter` ASC SEPARATOR '; ')
FROM `concat_table` WHERE `counter` > 10;
/* Выведет: 20; 30; 40 */
```

6.12. Переменные SQL

Результат текущего запроса можно сохранить в переменной и использовать в последующих запросах в рамках одного сеанса. Присвоить значение переменной можно следующими способами:

- с помощью оператора `SELECT`:

```
SELECT @time := NOW();
/* Выведет: 2018-02-12 16:08:34 */
SELECT @time;
/* Выведет: 2018-02-12 16:08:34 */
```

- с помощью оператора `SET`:

```
SET @time = NOW();
SELECT @time;
/* Выведет: 2018-02-12 16:09:44 */
```

Объявление переменной начинается с символа `@`, а сохранить значение в переменной позволяет оператор `:=`. Обратите внимание, что в случае применения оператора `SET` вместо оператора `:=` можно использовать оператор `=`.

В качестве примера создадим таблицу `var_table` в базе данных `tests`:

```
CREATE TABLE `var_table` (
   `id` INT NOT NULL AUTO_INCREMENT,
   `name_product` VARCHAR(255),
   `count` INT,
   PRIMARY KEY (`id`)
) ENGINE=MyISAM CHARSET=utf8 COLLATE utf8_general_ci;
```

Поле `name_product` предназначено для хранения названия товара, а поле `count` служит для обозначения количества товара на складе.

Добавим несколько записей:

```
INSERT INTO `var_table` VALUES (NULL, 'Монитор', 10);
INSERT INTO `var_table` VALUES (NULL, 'Клавиатура', 20);
INSERT INTO `var_table` VALUES (NULL, 'Мышь', 30);
INSERT INTO `var_table` VALUES (NULL, 'Тюнер', 40);
INSERT INTO `var_table` VALUES (NULL, 'HDD', 20);
```

Сохраним в переменной минимальное количество товара на складе, а затем выведем название товара с минимальным количеством:

```
SELECT @min := MIN(`count`) FROM `var_table`;
/* Выведет: 10 */
SELECT `name_product` FROM `var_table` WHERE `count` = @min;
/* Выведет: Монитор */
```

Если запрос вернет более одного варианта, то в переменной сохранится только последнее значение:

```
SELECT @min := `count` FROM `var_table`;
/* Выведет:
  10
  20
  30
  40
  20 */
SELECT @min;
/* Выведет: 20 */
```

6.13. Временные таблицы

Вре́менные таблицы создаются с помощью оператора `CREATE TEMPORARY TABLE`, синтаксис которого ничем не отличается от оператора `CREATE TABLE`. Временные таблицы используются для реализации дополнительного поиска в результатах выполнения запроса. Заполнить временную таблицу можно обычным способом, но чаще всего это делают с помощью вложенных запросов. Следует помнить, что имя временной таблицы действительно только в течение текущего соединения с сервером. После завершения соединения с сервером временная таблица автоматически удаляется.

В качестве примера создадим таблицу `user_table` в базе данных `tests`:

```
CREATE TABLE `user_table` (
   `id` INT NOT NULL AUTO_INCREMENT,
   `name` VARCHAR(255),
   PRIMARY KEY (`id`)
) ENGINE=MyISAM CHARSET=utf8 COLLATE utf8_general_ci;
```

В поле `name` мы будем хранить фамилию и имя пользователя. Добавим в таблицу несколько записей:

```
INSERT INTO `user_table` VALUES (NULL, 'Иванов Сергей');
INSERT INTO `user_table` VALUES (NULL, 'Иванов Николай');
INSERT INTO `user_table` VALUES (NULL, 'Иванов Иван');
INSERT INTO `user_table` VALUES (NULL, 'Петров Александр');
INSERT INTO `user_table` VALUES (NULL, 'Петров Николай');
INSERT INTO `user_table` VALUES (NULL, 'Иванов Максим');
```

А теперь инсценируем ситуацию поиска в найденном с помощью временных таблиц. Предположим, что первоначальный запрос пользователя выводит клиентов с фамилией `Иванов`:

```
SELECT `id`, `name` FROM `user_table` WHERE `name` LIKE '%Иванов%';
```

Сохраним результат запроса во временной таблице, а затем выведем клиентов только с именем `Николай`:

```
CREATE TEMPORARY TABLE `temp`
SELECT `id`, `name` FROM `user_table` WHERE `name` LIKE '%Иванов%';
SELECT `name` FROM `temp` WHERE `name` LIKE '%Николай%';
/* Выведет: Иванов Николай */
```

Обратите внимание: при использовании вложенных запросов не нужно определять структуру временной таблицы. По умолчанию структура временной таблицы будет такой же, как и в результирующей таблице, но без индексов. Посмотреть структуру временной таблицы можно с помощью оператора `EXPLAIN`:

```
CREATE TEMPORARY TABLE `temp2`
SELECT `id`, `name` FROM `user_table` WHERE `name` LIKE '%Иванов%';
EXPLAIN `temp2`;
```

Удалить временную таблицу можно следующими способами:

- с помощью оператора `DROP TABLE`:

  ```
  DROP TABLE <Имя временной таблицы>;
  ```

- по завершении соединения с сервером временная таблица будет удалена автоматически.

6.14. Вложенные запросы

Для изучения вложенных запросов создадим в базе данных `tests` следующие таблицы:

- `users_table` — для хранения данных о клиентах:

  ```
  CREATE TABLE `users_table` (
     `id_user` INT NOT NULL AUTO_INCREMENT,
     `name` VARCHAR(255),
     PRIMARY KEY (`id_user`)
  ) ENGINE=MyISAM CHARSET=utf8 COLLATE utf8_general_ci;
  ```

- `product_table` — для хранения данных о товарах:

  ```
  CREATE TABLE `product_table` (
     `id_product` INT NOT NULL AUTO_INCREMENT,
     `name_product` VARCHAR(255),
     PRIMARY KEY (`id_product`)
  ) ENGINE=MyISAM CHARSET=utf8 COLLATE utf8_general_ci;
  ```

- `orders_table` — для хранения сведений о покупках:

  ```
  CREATE TABLE `orders_table` (
     `id_order` INT NOT NULL AUTO_INCREMENT,
     `id_product` INT,
     `id_user` INT,
     `count` INT,
     PRIMARY KEY (`id_order`)
  ) ENGINE=MyISAM CHARSET=utf8 COLLATE utf8_general_ci;
  ```

Добавим в таблицы несколько записей:

```
INSERT INTO `users_table` VALUES (1, 'Иванов');
INSERT INTO `users_table` VALUES (2, 'Петров');

INSERT INTO `product_table` VALUES (1, 'Монитор');
INSERT INTO `product_table` VALUES (2, 'Клавиатура');
INSERT INTO `product_table` VALUES (3, 'Мышь');

INSERT INTO `orders_table` VALUES (1, 1, 1, 2);
INSERT INTO `orders_table` VALUES (2, 3, 2, 5);
INSERT INTO `orders_table` VALUES (3, 2, 1, 1);
```

6.14.1. Заполнение таблицы с помощью вложенного запроса

При изучении временных таблиц мы уже использовали вложенный запрос для заполнения временной таблицы. Заполнять с помощью вложенного запроса можно не только временные таблицы, но и обычные таблицы, создаваемые посредством оператора `CREATE TABLE`. Создадим таблицу `orders_item_table` с помощью вложенного запроса:

```
CREATE TABLE `orders_item_table` (
   `id_order` INT NOT NULL AUTO_INCREMENT,
   PRIMARY KEY (`id_order`)
) ENGINE=MyISAM CHARSET=utf8 COLLATE utf8_general_ci
SELECT `orders_table`.`id_order` AS `id_order`,
`users_table`.`name` AS `user`,
`product_table`.`name_product` AS `name`,
`orders_table`.`count` AS `count`
FROM `users_table`, `product_table`, `orders_table`
WHERE `orders_table`.`id_user` = `users_table`.`id_user` AND
`orders_table`.`id_product` = `product_table`.`id_product`;
```

Выведем структуру созданной таблицы с помощью SQL-запроса:

```
EXPLAIN `orders_item_table`\G
```

Эта команда SQL выведет:

```
*************************** 1. row ***************************
Field: id_order
Type: int(11)
Null: NO
Key: PRI
Default: NULL
Extra: auto_increment
*************************** 2. row ***************************
Field: user
Type: varchar(255)
Null: YES
Key:
Default: NULL
Extra:
*************************** 3. row ***************************
Field: name
Type: varchar(255)
Null: YES
Key:
Default: NULL
Extra:
```

```
*************************** 4. row ***************************
Field: count
Type: int(11)
Null: YES
Key:
Default: NULL
Extra:
```

Как видно из результата, столбцы, не определенные в операторе `CREATE TABLE`, но имеющиеся в результирующей таблице, добавляются в новую таблицу. Если столбцы определены в операторе `CREATE TABLE`, но отсутствуют во вложенном запросе, то они получают значение по умолчанию.

Использовать вложенные запросы можно и в операторе `INSERT`. Создадим таблицу `orders_item2_table` обычным образом:

```
CREATE TABLE `orders_item2_table` (
   `id_order` INT NOT NULL AUTO_INCREMENT,
   `user` VARCHAR(255),
   `name` VARCHAR(255),
   `count` INT,
   PRIMARY KEY (`id_order`)
) ENGINE=MyISAM CHARSET=utf8 COLLATE utf8_general_ci;
```

Затем добавим записи с помощью оператора `INSERT` и вложенного запроса:

```
INSERT IGNORE INTO `orders_item2_table`
SELECT `orders_table`.`id_order` AS `id_order`,
`users_table`.`name` AS `user`,
`product_table`.`name_product` AS `name`,
`orders_table`.`count` AS `count`
FROM `users_table`, `product_table`, `orders_table`
WHERE `orders_table`.`id_user` = `users_table`.`id_user` AND
`orders_table`.`id_product` = `product_table`.`id_product`;
```

Ключевое слово `IGNORE` сообщает серверу, что записи с повторяющимися значениями первичного ключа и уникальных индексов должны отбрасываться. Если не указать это ключевое слово, то при попытке вставить в таблицу повторяющуюся запись возникнет ошибка.

Если мы хотим, чтобы записи с повторяющимися значениями ключа заменяли собой уже существующие в таблице записи, то применим оператор `REPLACE`:

```
REPLACE INTO `orders_item2_table`
SELECT `orders_table`.`id_order` AS `id_order`,
`users_table`.`name` AS `user`,
`product_table`.`name_product` AS `name`,
`orders_table`.`count` AS `count`
FROM `users_table`, `product_table`, `orders_table`
WHERE `orders_table`.`id_user` = `users_table`.`id_user` AND
`orders_table`.`id_product` = `product_table`.`id_product`;
```

6.14.2. Применение вложенных запросов в инструкции *WHERE*

Выведем имя пользователя, сделавшего заказ под номером 2, с помощью вложенного запроса:

```
SELECT `name` FROM `users_table`
WHERE `id_user` = (SELECT `id_user` FROM `orders_table`
WHERE `id_order` = 2);
/* Выведет: Петров */
```

В этом примере мы объединили два запроса в один: внутренний запрос возвращает идентификатор пользователя, сделавшего заказ с номером 2, а внешний запрос по этому идентификатору получает имя пользователя. Как видно из примера, вложенный запрос всегда заключается в круглые скобки.

Уровень вложенности запроса может быть более двух. Однако на практике запросы с уровнем вложенности более трех нецелесообразны, т. к. это приводит к увеличению времени выполнения запроса.

Если вложенный запрос возвращает более одного значения, то MySQL генерирует ошибку. Обойти эту проблему можно следующими способами:

- указать ключевые слова `IN` или `NOT IN`:

  ```
  SELECT `name` FROM `users_table`
  WHERE `id_user` IN (SELECT `id_user` FROM `orders_table`);
  ```

  Этот пример выведет:

  ```
  Иванов
  Петров
  ```

- использовать ключевые слова `ANY` или `SOME`:

  ```
  SELECT `name` FROM `users_table`
  WHERE `id_user` > ANY (SELECT `id_user` FROM `orders_table`);
  /* Выведет: Петров */
  ```

- задать ключевое слово `ALL`:

  ```
  SELECT `name` FROM `users_table`
  WHERE `id_user` <= ALL (SELECT `id_user` FROM `orders_table`);
  /* Выведет: Иванов */
  ```

При указании ключевого слова `IN` проверяется совпадение с одним из значений, возвращаемых вложенным запросом. При использовании ключевого слова `NOT IN`, наоборот, проверяется отсутствие совпадения со списком значений. Если применяется ключевые слова `ANY` или `SOME`, то проверяемое значение поочередно сравнивается с каждым элементом, и если хотя бы одно сравнение возвращает значение `Истина`, то результат попадает в итоговую таблицу. Если задано ключевое слово `ALL`, то в результирующую таблицу попадут значения, только если все сравнения вернут значение `Истина`.

С помощью ключевого слова `EXISTS` можно проверить, имеется ли в результирующей таблице хоть одна строка. Если вложенный запрос дает непустой результат, то ключевое слово `EXISTS` возвращает `1` (истина). В противном случае возвращается значение `0` (ложь). Получить противоположные значения позволяет ключевое слово `NOT EXISTS`.

В качестве примера выведем фамилии всех клиентов, сделавших хотя бы один заказ. Для наглядности добавим в таблицу `users_table` еще одного клиента:

```
INSERT INTO `users_table` VALUES (3, 'Сидоров');
```

Теперь выполним такой запрос:

```
SELECT `name` FROM `users_table`
WHERE EXISTS (SELECT * FROM `orders_table`
WHERE `orders_table`.`id_user` = `users_table`.`id_user`);
```

В результате мы получим:

```
Иванов
Петров
```

А теперь выведем фамилии клиентов, не сделавших ни одного заказа:

```
SELECT `name` FROM `users_table`
WHERE NOT EXISTS (SELECT * FROM `orders_table`
WHERE `orders_table`.`id_user` = `users_table`.`id_user`);
```

Этот запрос вернет:

```
Сидоров
```

Обратите внимание: внутри вложенного запроса мы указываем поле таблицы `` `users_table`.`id_user` `` из внешнего запроса. Такая связь называется *внешней ссылкой*, а сам запрос называется *коррелированным вложенным запросом*.

6.14.3. Применение вложенных запросов в инструкции *FROM*

Вложенные запросы можно использовать и в инструкции `FROM`. При этом мы фактически выполним выборку данных, возвращенных вложенным запросом.

Для примера давайте рассмотрим созданную в *разд. 6.13* таблицу `user_table`, хранящую список пользователей. Ранее для выполнения сложного поиска мы сначала перенесли предварительно отобранные записи во временную таблицу, в которой потом выполняли окончательный поиск. Но ту же операцию можно реализовать гораздо проще — указав в инструкции `FROM` вложенный запрос:

```
SELECT `name` FROM
(SELECT `name` FROM `user_table` WHERE `name` LIKE '%Иванов%') AS `users`
WHERE `name` LIKE '%Николай%';
/* Выведет: Иванов Николай */
```

Здесь вложенный запрос выполняет предварительную выборку записей, включающих слово `Иванов`. Основной запрос уже производит окончательный поиск записей, найденных вложенным запросом и хранящих слово `Николай`.

Отметим, что для вложенного запроса всегда следует указывать псевдоним, применив инструкцию `AS`.

6.15. Внешние ключи

При эксплуатации реляционной базы данных время от времени необходимо изменять ее структуру или удалять устаревшие данные. Например, для увеличения быстродействия можно удалить учетные записи клиентов, которые не совершали покупок в течение определенного срока. Если просто удалить этих клиентов из одной таблицы, то это может привести к нарушению ссылочной целостности базы данных, т. к. кто-нибудь из удаляемых клиентов наверняка совершал ранее покупки, а значит, сведения о покупке заносились в таблицу заказов. По этой причине при удалении клиента необходимо предварительно изъять все записи о совершенных им покупках из таблицы заказов. Сделать это можно с помощью двух SQL-запросов. Первый запрос удаляет записи из таблицы заказов, а второй — удаляет запись о клиенте.

Для таблиц типа `InnoDB` предусмотрена возможность автоматического контроля над ссылочной целостностью базы данных с помощью *внешних ключей*.

Внешний ключ указывает, что поле или комбинация полей текущей таблицы содержат ссылку на другую таблицу. Его можно добавить при создании таблицы с помощью оператора `CREATE TABLE`, а оператор `ALTER TABLE` позволяет добавить внешний ключ в уже существующую таблицу.

Добавляется внешний ключ с помощью конструкции `FOREIGN KEY`. Формат конструкции:

```
[CONSTRAINT [<Имя>] ]
FOREIGN KEY [<Имя ключа>] (<Список полей в текущей таблице>)
REFERENCES <Имя внешней таблицы> (<Список полей во внешней таблице>)
[ON DELETE <Действие>]
[ON UPDATE <Действие>]
```

В параметре `<Действие>` могут быть указаны значения:

- `CASCADE` — удаление или изменение записи, содержащей первичный ключ, приведет к автоматическому удалению или изменению соответствующих записей в таблице-потомке;
- `SET NULL` — при удалении или изменении записи, содержащей первичный ключ, соответствующие записи в таблице-потомке получат значение `NULL`;
- `RESTRICT` — нельзя удалить или изменить запись, пока в таблице-потомке существуют ссылающиеся записи;
- `NO ACTION` — то же самое, что и `RESTRICT`.

Если действие не указано, это равносильно указанию действия `RESTRICT`.

Для примера создадим в базе данных `tests` следующие таблицы:

- `users_foreign` — для хранения данных о клиентах:

```
CREATE TABLE `users_foreign` (
   `id_user` INT NOT NULL AUTO_INCREMENT,
   `name` VARCHAR(255),
   PRIMARY KEY (`id_user`)
) ENGINE=InnoDB CHARSET=utf8 COLLATE utf8_general_ci;
```

- `product_foreign` — для хранения данных о товарах:

```
CREATE TABLE `product_foreign` (
   `id_product` INT NOT NULL AUTO_INCREMENT,
   `name_product` VARCHAR(255),
   PRIMARY KEY (`id_product`)
) ENGINE=InnoDB CHARSET=utf8 COLLATE utf8_general_ci;
```

- `orders_foreign` — для хранения сведений о покупках:

```
CREATE TABLE `orders_foreign` (
   `id_order` INT NOT NULL AUTO_INCREMENT,
   `id_product` INT,
   `id_user` INT,
   `count` INT,
   PRIMARY KEY (`id_order`),
   FOREIGN KEY (`id_user`) REFERENCES `users_foreign` (`id_user`)
   ON DELETE CASCADE ON UPDATE CASCADE,
   FOREIGN KEY (`id_product`)
   REFERENCES `product_foreign` (`id_product`)
   ON DELETE RESTRICT ON UPDATE RESTRICT
) ENGINE=InnoDB CHARSET=utf8 COLLATE utf8_general_ci;
```

Добавим в таблицы несколько записей:

```
INSERT INTO `users_foreign` VALUES (1, 'Иванов');
INSERT INTO `users_foreign` VALUES (2, 'Петров');

INSERT INTO `product_foreign` VALUES (1, 'Монитор');
INSERT INTO `product_foreign` VALUES (2, 'Клавиатура');
INSERT INTO `product_foreign` VALUES (3, 'Мышь');

INSERT INTO `orders_foreign` VALUES (1, 1, 1, 2);
INSERT INTO `orders_foreign` VALUES (2, 3, 2, 5);
INSERT INTO `orders_foreign` VALUES (3, 2, 1, 1);
```

Теперь попробуем удалить господина Иванова из таблицы `users_foreign`:

```
DELETE FROM `users_foreign` WHERE `id_user`=1;
SELECT `name` FROM `users_foreign`;
/* Выведет: Петров */
```

Посмотрим, сколько заказов осталось в таблице `orders_foreign`:

```
SELECT COUNT(`id_order`) FROM `orders_foreign`;
/* Выведет: 1 */
```

Как видно из этого примера, удаление господина Иванова привело к автоматическому удалению его заказов за счет применения ключевого слова `CASCADE`. Теперь попробуем добавить заказ на имя уже не существующего в базе данных господина Иванова:

```
INSERT INTO `orders_foreign` VALUES (NULL, 1, 1, 2);
```

В итоге получим ошибку:

```
ERROR 1452 (23000): Cannot add or update a child row: a foreign key
constraint fails
```

Попробуем удалить товар с номером `3` из таблицы `product_foreign`:

```
DELETE FROM `product_foreign` WHERE `id_product`=3;
```

В итоге также получим ошибку:

```
ERROR 1451 (23000): Cannot delete or update a parent row: a foreign key
constraint fails
```

Иными словами, пока мы не удалим заказ с номером `2` из таблицы `orders_foreign`, мы не сможем удалить товар с номером `3` из таблицы `product_foreign`. Это достигается за счет применения ключевого слова `RESTRICT`.



ГЛАВА 7

AJAX. Обмен данными без перезагрузки Web-страницы

7.1. Подготовка к загрузке данных

AJAX (Asynchronous JavaScript and XML, асинхронный JavaScript и XML) — это технология программной подгрузки произвольных данных для их вывода на страницу (возможно, после обработки) или для использования в вычислениях. Программа инициирует загрузку файла с данными, а потом считывает его содержимое и пускает в обработку, — и все это *без перезагрузки* самой страницы.

Так можно загружать данные трех различных типов:

- фрагменты HTML-кода или обычного текста, которые мы можем просто вывести на экран, вставив их в любой контейнер;
- данные, закодированные с помощью языка XML. Понятно, что просто вывести их на экран нельзя, и нам потребуется раскодировать их и преобразовать в подходящий для вывода в составе страницы формат — HTML-код;
- данные, закодированные в формате JSON (о нем речь пойдет чуть позже). Их также потребуется раскодировать и преобразовать в HTML-код.

> ***Внимание!***
> Технология AJAX позволяет загружать данные исключительно с Web-сервера. Загрузка из локальных файлов во всех Web-браузерах заблокирована в целях безопасности.

Прежде чем загрузить какой-либо файл с применением технологии AJAX, нам следует получить объект, который, собственно, и выполнит его загрузку. Процесс его получения различается в зависимости от используемого Web-браузера.

7.1.1. Стандартный способ

В Firefox, Chrome, Opera, Safari и Internet Explorer, начиная с его версии 7, загрузкой данных «заведует» класс `XMLHttpRequest`. Этот класс поддерживается самим Web-браузером и определен в стандарте DOM, поэтому способ загрузки данных с его помощью носит название *стандартного*.

Нам нужно лишь создать объект упомянутого класса оператором `new`. Никакие параметры при этом не указываются:

```
var oAJAX = new XMLHttpRequest();
```

Класс `XMLHttpRequest` также доступен через одноименное свойство объекта `window`.

7.1.2. Способ, применяемый в Internet Explorer 5 и 6

Однако в Internet Explorer версий 5 и 6 класс `XMLHttpRequest` не поддерживается. Вместо этого следует использовать полностью аналогичный класс `Microsoft.XMLHTTP`.

Пример создания объекта:

```
var oAJAX = new ActiveXObject("Microsoft.XMLHTTP");
```

> **Внимание!**
> В дальнейшем, описывая реализацию технологии AJAX и говоря о классе `XMLHttpRequest`, мы будем иметь в виду также и класс `Microsoft.XMLHTTP`, поскольку они полностью идентичны по своим возможностям.

7.1.3. Универсальный способ

На практике нам будет полезнее универсальный, кроссплатформенный способ, одинаково работающий во всех Web-браузерах. Код, реализующий этот способ, приведен в листинге 7.1.

Листинг 7.1. Кроссплатформенная загрузка данных

```
if (window.XMLHttpRequest) {
   var oAJAX = new XMLHttpRequest();
}
else {
   var oAJAX = new ActiveXObject("Microsoft.XMLHTTP");
}
```

7.2. Отправка запроса

Получив объект, реализующий технологию AJAX, мы можем отправить Web-серверу запрос на загрузку файла с данными.

7.2.1. Синхронный или асинхронный запрос?

Существуют две разновидности запросов на получение данных AJAX, которые мы можем отправить:

- *синхронный запрос*, при котором Web-браузер приостанавливает выполнение программы и ждет, пока файл с данными не будет получен;

- *асинхронный запрос*, при котором Web-браузер продолжает выполнять программу, не дожидаясь получения запрошенного файла, а когда он, наконец, будет получен, генерирует особое событие (о нем мы поговорим далее).

На практике чаще используются асинхронные запросы, т. к. они позволяют получить данные по ходу выполнения прочего JavaScript-кода и, соответственно, не приводят к «зависанию» всей страницы.

7.2.2. Задание параметров запроса

Сначала нам следует указать параметры отправляемого запроса: метод отсылки данных (`GET` или `POST`), URL-адрес файла с данными или программы, которая сгенерирует эти данные, и вид запроса (синхронный или асинхронный). Все это выполняет метод `open()` класса `XMLHttpRequest`:

```
open(<Метод отправки данных>, <URL-адрес>, <true | false>)
```

`<Метод отправки данных>` указывается в виде строки `"GET"` или `"POST"`. `<URL-адрес>`, с которого запрашиваются данные, также указывается как строка. Если третьим параметром передано значение `true`, будет выполнен асинхронный запрос, если `false` — синхронный. Метод `open()` не возвращает результат.

Приведем пару примеров:

- задаем параметры *синхронного* запроса на получение файла `fragment.html` (находится в корневом каталоге сервера), в котором хранится фрагмент HTML-кода для вывода на страницу:

  ```
  oAJAX.open("GET", "/fragment.html", false);
  ```

- а здесь мы задаем параметры для *асинхронного* запроса на получение данных от программы `search.php`, причем входные данные для этой программы (они могут быть введены посетителем в специально предусмотренной для этого форме или сгенерированы скриптом) мы отошлем методом `POST`:

  ```
  oAJAX.open("POST", "search.php", true);
  ```

7.2.3. Задание MIME-типа отправляемых данных

Если данные, которые мы собираемся подгрузить, генерируются выполняющейся на стороне сервера программой, она для работы может требовать какую-либо входную информацию. Эта информация может как вводиться посетителем в специальной форме, так и генерироваться скриптом. И отправить мы ее можем методом `GET` или `POST`.

Если мы отправляем входные данные методом `POST`, то обязаны указать соответствующий MIME-тип этих данных с помощью метода `setRequestHeader()` класса `XMLHttpRequest`:

```
setRequestHeader(<Заголовок>, <Значение>)
```

Оба параметра этого метода указываются в виде строк. В нашем случае именем параметра станет строка "Content-Type" (именно этот параметр указывает MIME-тип данных), а его значением — наименование нужного метода отправки данных:

```
oAJAX.setRequestHeader("Content-Type",
                        "application/x-www-form-urlencoded");
```

Метод setRequestHeader() не возвращает результат.

7.2.4. Собственно отправка запроса

Вот теперь мы можем, наконец, отправить запрос. Выполняется он вызовом не возвращающего результат метода send() класса XMLHttpRequest:

```
oAJAX.send();
```

7.2.5. Отправка данных с запросом

Но как же отправить входную информацию для серверной программы, которая сгенерирует нам данные? Это зависит от метода отсылки входной информации, что мы указали первым параметром метода open().

Если входные данные отправляются методом GET, они просто добавляются к URL-адресу, указанному вторым параметром метода open(). Тогда метод send() вызывается без указания параметров.

Если входные данные отправляются методом POST, эти данные указываются в качестве единственного параметра метода send(). К запрашиваемому URL-адресу они в этом случае не добавляются.

В качестве примера рассмотрим форму входа на сайт:

```
<form id="frmLogin" action="login.php"
      enctype="application/x-www-form-urlencoded">
  <p>Имя: <input type="text" name="login"></p>
  <p>Пароль: <input type="password" name="password"></p>
  <p><input type="submit" name="submit" value="Войти"></p>
</form>
```

По нажатию кнопки **Войти** будет выполнен следующий скрипт, который создаст строку sData с введенными в форму данными, закодированными соответствующим образом:

```
var oForm = document.getElementById("frmLogin");
var login = oForm.elements["login"].value;
var password = oForm.elements["password"].value;
var sData = "login=" + encodeURIComponent(login) + "&";
sData += "password=" + encodeURIComponent(password);
```

Тогда, чтобы отправить эти данные серверной программе методом GET, мы напишем такой скрипт:

```
oAJAX.open("GET", oForm.action + "?" + sData, true);
oAJAX.send();
```

А вот скрипт для отправки данных методом `POST`:

```
oAJAX.open("POST", oForm.action, true);
oAJAX.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
oAJAX.send(sData);
```

7.3. Получение данных

Получив от нас AJAX-запрос на подгрузку данных, Web-сервер либо отправит нам запрошенный файл, либо вызовет серверную программу, которая получит отправленную нами входную информацию и сгенерирует результирующие данные, которые, опять же, будут отправлены нам. Осталась мелочь — получить и обработать их.

7.3.1. Назначение обработчика изменения статуса

Сначала нам следует написать код, который будет обрабатывать полученные данные, — иначе говоря, обработчик данных. Этот код можно оформить двумя способами.

Обработчик в случае *асинхронного* запроса можно оформить в виде функции (обычной или анонимной), которую нужно присвоить свойству `onreadystatechange` класса `XMLHttpRequest`. Эта функция будет вызываться несколько раз при каждом изменении статуса обработки запроса.

> ***Внимание!***
>
> Присвоение функции-обработчика свойству `onreadystatechange` следует выполнять перед вызовом метода `send()`. Существует вероятность того, что запрошенные данные загрузятся ранее, чем для них будет указан обработчик, и такую ситуацию лучше исключить.

Пример:

```
oAJAX.open("GET", "search.php?s=test", true);
oAJAX.onreadystatechange = function() {
   getData(oAJAX);
};
oAJAX.send();
```

Содержимое функции `getData()`:

```
function getData(obj) {
   if ((obj.readyState == 4) && (obj.status == 200)) {
      console.log(obj.responseText);
   }
}
```

Если был выполнен *синхронный* запрос, код обработчика можно просто поместить после вызова метода `send()`:

```
oAJAX.open("GET", "search.php?s=test", false);
oAJAX.send();
if ((oAJAX.readyState == 4) && (oAJAX.status == 200)) {
   console.log(oAJAX.responseText);
}
```

7.3.2. Определение успешного получения данных

Если мы используем первый способ оформления кода, который будет обрабатывать полученные данные, то должны приготовиться к тому, что событие `onreadystatechange` будет возникать всякий раз, когда изменяется состояние ожидания этих данных (когда запрос собственно отправляется, когда данные получены, но еще не обработаны, и др.). И еще нам следует иметь в виду, что Web-сервер или серверная программа могут вернуть не только запрашиваемые нами данные, но и сообщение об ошибке.

В рассматриваемом случае нам понадобятся следующие свойства класса `XMLHttpRequest`:

- `readyState` — возвращает число, обозначающее состояние ожидания данных:
  - `0` — соединение с Web-сервером еще не установлено;
  - `1` — соединение с Web-сервером установлено;
  - `2` — вызван метод `send()`, доступны заголовки и статус;
  - `3` — идет обработка загруженных данных;
  - `4` — данные обработаны и могут быть извлечены;
- `status` — возвращает код ответа Web-сервера в виде числа, например, `200` (данные успешно получены) или `404` (файл не найден).

Пример:

```
oAJAX.open("GET", "search.php?s=test", true);
oAJAX.onreadystatechange = function() {
   if ((oAJAX.readyState == 4) && (oAJAX.status == 200)) {
      console.log(oAJAX.responseText);
   }
};
oAJAX.send();
```

7.3.3. Собственно получение данных

Загруженные данные мы можем извлечь из свойства `responseText` класса `XMLHttpRequest`. Эти данные представляют собой строку, хранящую фрагмент HTML-кода или данные, закодированные в форматах XML или JSON.

В листинге 7.2 приведен полный код, подгружающий фрагмент HTML-кода, возвращаемого скриптом search.php (листинг 7.3), и выводящий его на страницу в кон-

тейнере `output`. По нажатию кнопки **Выполнить запрос** мы будем отправлять число миллисекунд, прошедшее с 1 января 1970 г., и получать обработанные данные.

Листинг 7.2. Пример загрузки с сервера фрагмента HTML-кода

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>Пример загрузки с сервера фрагмента HTML-кода</title>
<script>
function handler() {
   var oAJAX = null;
   if (window.XMLHttpRequest) {
      oAJAX = new XMLHttpRequest();
   }
   else {
      oAJAX = new ActiveXObject("Microsoft.XMLHTTP");
   }
   if (!oAJAX) return;
   oAJAX.open("GET", "search.php?s=" + Date.now(), true);
   oAJAX.onreadystatechange = function() {
      if (oAJAX.readyState == 4) {
         var oOutput = document.getElementById("output");
         if (oAJAX.status == 200) {
            oOutput.innerHTML = oAJAX.responseText;
         }
         else {
            oOutput.innerHTML = "Ошибка";
         }
      }
   };
   oAJAX.send();
}
</script>
</head>
<body>
   <div id="output"></div>
   <input type="button" value="Выполнить запрос" onclick="handler()">
</body>
</html>
```

Листинг 7.3. Содержимое файла search.php

```
<?php
if (isset($_GET['s'])) {
   $s = htmlspecialchars($_GET['s'], ENT_COMPAT | ENT_HTML5, 'UTF-8');
```

```
   echo 'Получены данные: <b>' . $s . '</b>';
}
else {
   echo 'Нет данных';
}
```

7.4. Формат JSON

Ранее мы говорили, что данные, предназначенные для дальнейшей обработки, можно кодировать в формате JSON (JavaScript Object Notation, объектная нотация JavaScript). Во многих случаях этот формат намного удобнее, чем XML, из-за его компактности и простоты обработки.

7.4.1. Описание формата JSON

Данные, закодированные в формате JSON, представляют собой строку с кодом, объявляющим объект класса `Object`. В свойствах этого объекта, собственно, и хранятся отдельные значения, составляющие массив закодированных данных.

Здесь нужно помнить три момента:

- в данных JSON имена свойств также берутся в кавычки;
- значениями свойств не могут быть функции — т. е. в объектах, закодированных в JSON, могут присутствовать лишь свойства, но никак не методы;
- данные JSON всегда сохраняются в кодировке UTF-8.

Приведем пару примеров:

- преобразование данных JSON в объект:

```
var data = '{ "id": "3.20.4", "title": "Формат JSON" }';
var obj = JSON.parse(data);
console.log(obj.id);
console.log(obj.title);
```

- кодирование целого массива данных:

```
var data = '{ "status": 1, "data": [ {"id": 1}, {"id": 2} ] }';
var obj = JSON.parse(data);
console.log(obj.status);
console.log(obj.data[0].id);
console.log(obj.data[1].id);
```

  Здесь свойство `status` хранит числовой код состояния (`1` обозначает отсутствие ошибок при генерировании данных серверной программой), а свойство `data` — массив с данными.

7.4.2. Декодирование данных JSON: стандартный способ

В новых Web-браузерах язык JavaScript поддерживает объект `JSON`. Метод `parse()` этого объекта выполняет декодирование данных JSON:

```
JSON.parse(<Данные JSON>)
```

Данные JSON передаются в виде строки (собственно, в виде строки мы и получим их из свойства `responseText` класса `XMLHttpRequest`). Метод возвращает сгенерированный на основе этих данных объект класса `Object`:

```
try {
   var oData = JSON.parse(oAJAX.responseText);
   console.log(oData);
} catch (e) {
   console.log("Не удалось декодировать");
}
```

7.4.3. Декодирование данных JSON: способ, применяемый в устаревших Web-браузерах

Старые Web-браузеры, в частности Internet Explorer 7 и более ранние его версии, не поддерживают объект `JSON`. Поэтому в них для декодирования данных JSON нам придется применять функцию `eval()`. Для корректной работы функции `eval()` строку, содержащую объект JSON, необходимо взять в круглые скобки:

```
try {
   var oData = eval('(' + oAJAX.responseText + ')');
   console.log(oData);
} catch (e) {
   console.log("Не удалось декодировать");
}
```

7.4.4. Декодирование данных JSON: универсальный способ

Чтобы скрипты, загружающие и обрабатывающие данные JSON, работали во всех Web-браузерах, мы применим универсальный способ, написав следующий код:

```
var oData = null;
if (JSON) {
   oData = JSON.parse(oAJAX.responseText);
}
else {
   oData = eval('(' + oAJAX.responseText + ')');
}
```

В листинге 7.4 приведен полный код, подгружающий данные в формате JSON, возвращаемые скриптом search.php (листинг 7.5), и выводящий их на страницу в контейнере `output`. По нажатию кнопки **Выполнить запрос** мы будем отправлять число миллисекунд, прошедшее с 1 января 1970 г., и получать обработанные данные в формате JSON.

Листинг 7.4. Пример загрузки и вывода данных

```
<!DOCTYPE html>
<html lang="ru">
<head>
   <meta charset="utf-8">
   <title>AJAX и JSON</title>
<script>
function handler() {
   var oAJAX = null;
   if (window.XMLHttpRequest) {
      oAJAX = new XMLHttpRequest();
   }
   else {
      oAJAX = new ActiveXObject("Microsoft.XMLHTTP");
   }
   if (!oAJAX) return;
   oAJAX.open("GET", "search.php?s=" + Date.now(), true);
   oAJAX.onreadystatechange = function() {
      if (oAJAX.readyState == 4) {
         var oOutput = document.getElementById("output");
         if (oAJAX.status == 200) {
            try {
               var oData = null;
               if (JSON) {
                  oData = JSON.parse(oAJAX.responseText);
               }
               else {
                  oData = eval('(' + oAJAX.responseText + ')');
               }
               if (oData.status == 1) {
                  oOutput.textContent = oData.data;
               }
               else {
                  oOutput.textContent = "Получены ошибочные данные";
               }
            } catch (e) {
               oOutput.textContent = "Не удалось декодировать";
            }
         }
```

```
        else {
            oOutput.innerHTML = "Ошибка получения данных";
        }
      }
   };
   oAJAX.send();
}
</script>
</head>
<body>
   <div id="output"></div>
   <input type="button" value="Выполнить запрос" onclick="handler()">
</body>
</html>
```

Листинг 7.5. Содержимое файла search.php

```
<?php
// Запрет кэширования
header('Expires: Wed, 18 Oct 2017 23:17:32 GMT');
header('Cache-Control: no-store, no-cache, must-revalidate');
header('Pragma: no-cache');
// MIME-тип ответа сервера
header('Content-Type: application/json; charset=utf-8');
if (isset($_GET['s'])) {
   $s = htmlspecialchars($_GET['s'], ENT_COMPAT | ENT_HTML5, 'UTF-8');
   echo '{ "status": 1, "data": "' . $s. '" }';
}
else {
   echo '{ "status": 0, "data": "Нет данных" }';
}
```

7.4.5. Преобразование объекта в строку в формате JSON

Преобразовать объект JavaScript в строку в формате JSON позволяет метод `stringify()`. Формат метода:

```
JSON.stringify(<Объект>[, <Функция или массив>[, <Формат>]])
```

В первом параметре задается объект, преобразуемый в строку в формате JSON. Во втором параметре можно указать массив с именами свойств, подлежащих преобразованию, или функцию, в которую будут передаваться два параметра: название свойства и его значение. Внутри функции нужно вернуть новое значение. Если функция возвращает значение `undefined`, то свойство не попадет в итоговую строку. По умолчанию в строке отсутствуют какие-либо разделители между свойствами и значениями. Этот разделитель можно указать в параметре `<Формат>`:

```
var obj = { "id": "3.20.4", "title": "Формат JSON" };
console.log(JSON.stringify(obj));
// {"id":"3.20.4","title":"Формат JSON"}
console.log(JSON.stringify(["Один", "Два"]));
// ["Один","Два"]
console.log(JSON.stringify(obj, ["id"], " "));
// {
//  "id": "3.20.4"
// }
console.log(JSON.stringify(obj, function(key, value) {
   if (key === "id") {
      return undefined;
   }
   return value;
}));
// {"title":"Формат JSON"}
```

7.4.6. Кодирование и декодирование данных в формате JSON в PHP

Создать строку в формате JSON на основе массива в PHP позволяет функция `json_encode()`:

```
string json_encode( mixed $value [, int $options = 0 [,
                    int $depth = 512 ]] )
```

В первом параметре указывается значение, подлежащее преобразованию, — например, обычный или ассоциативный массив. Все строковые значения, присутствующие в кодируемом массиве, должны храниться в кодировке UTF-8. Во втором параметре можно задать различные опции, а в третьем — максимальную глубину. Функция `json_encode()` возвращает строку с закодированными данными или `false`, если при кодировании произошла ошибка:

```
$data = [
   "status" => 1,
   "data" => "Данные"];
echo json_encode($data);
// {"status":1,"data":"\u0414\u0430\u043d\u043d\u044b\u0435"}
echo json_encode($data, JSON_UNESCAPED_UNICODE);
// {"status":1,"data":"Данные"}
```

Во втором параметре можно указать следующие опции или их комбинацию через оператор `|` (полный список опций смотрите в документации):

- `JSON_HEX_TAG` — преобразовывать символы `<` и `>` в последовательности `\u003C` и `\u003E`;
- `JSON_HEX_AMP` — преобразовывать символы амперсанда `&` в последовательности `\u0026`;

- `JSON_HEX_APOS` — преобразовывать символы апострофа `'` в последовательности `\u0027`;
- `JSON_HEX_QUOT` — преобразовывать символы двойных кавычек `"` в последовательности `\u0022`;
- `JSON_FORCE_OBJECT` — при кодировании списка возвращать объект, а не массив;
- `JSON_NUMERIC_CHECK` — кодировать строки, содержащие числа, как числа;
- `JSON_PRETTY_PRINT` — посредством пробелов форматировать возвращаемые данные для удобства чтения;
- `JSON_UNESCAPED_SLASHES` — не экранировать символы слеша;
- `JSON_UNESCAPED_UNICODE` — не кодировать многобайтовые символы (по умолчанию они преобразуются в последовательности вида `\u<Код символа>`).

Вот пример указания нескольких опций:

```
$data = [
   "status" => 1,
   "data" => "Данные"];
echo json_encode($data, JSON_PRETTY_PRINT | JSON_UNESCAPED_UNICODE);
/* {
   "status": 1,
   "data": "Данные"
} */
```

Выполнить обратное преобразование позволяет функция `json_decode()`:

```
mixed json_decode( string $json [, bool $assoc = false [,
                 int $depth = 512 [, int $options = 0 ]]] )
```

В первом параметре задается строка в формате JSON. Если во втором параметре указано значение `true`, то результатом станет ассоциативный массив. Третий параметр задает максимальную глубину, а четвертый — дополнительные опции. Функция возвращает преобразованные данные или значение `null` в случае ошибки:

```
$json = '{"status":1,"data":"Данные"}';
print_r(json_decode($json));
/* (
   [status] => 1
   [data] => Данные
) */
```

Выяснить, какого рода ошибка возникла при кодировании или декодировании данных, можно вызовом функции `json_last_error()`. Она вернет одно из значений следующих констант (полный их список смотрите в документации):

- `JSON_ERROR_NONE` — никаких ошибок не возникло;
- `JSON_ERROR_DEPTH` — превышена максимальная глубина;
- `JSON_ERROR_STATE_MISMATCH` — некорректная структура данных;

- `JSON_ERROR_CTRL_CHAR` — некорректный управляющий символ или неверная кодировка;
- `JSON_ERROR_SYNTAX` — синтаксическая ошибка;
- `JSON_ERROR_UTF8` — некорректный символ UTF-8 или неверная кодировка.

Теперь еще два момента:

- во-первых, нам следует задать в качестве MIME-типа отправляемых данных `application/json`, поставив в самое начало скрипта строку:

  ```
  header('Content-Type: application/json; charset=utf-8');
  ```

- во-вторых, настоятельно рекомендуется отключить кэширование отправляемых данных на стороне клиента.

В качестве примера изменим код, который приведен в листинге 7.5 (листинг 7.6).

Листинг 7.6. Генерирование данных JSON

```
<?php
// Запрет кэширования
header('Expires: Wed, 18 Oct 2017 23:17:32 GMT');
header('Cache-Control: no-store, no-cache, must-revalidate');
header('Pragma: no-cache');
// MIME-тип ответа сервера
header('Content-Type: application/json; charset=utf-8');
if (isset($_GET['s'])) {
   $s = htmlspecialchars($_GET['s'], ENT_COMPAT | ENT_HTML5,
                         'UTF-8');
   $data = [ "status" => 1, "data" => $s];
   $json = json_encode($data);
   if ($json !== false) {
      echo $json;
   }
   else {
      echo json_encode([ "status" => 0 ]);
   }
}
else {
   echo json_encode([ "status" => 0 ]);
}
```

Если в процессе кодирования данных произошла ошибка, мы отправляем клиентскому скрипту массив с элементом `status`, имеющим значение `0`. Тем самым мы сообщим ему об ошибке.

Не забываем, что данные, имеющие формат JSON, должны быть в кодировке UTF-8. В связи с этим нам придется сохранить в кодировке UTF-8 (обязательно без BOM!) и сами страницы, выводящие данные JSON на экран, и PHP-скрипты, генерирующие эти данные.

ПРИЛОЖЕНИЕ

Описание электронного архива

Электронный архив с материалами, сопровождающими книгу, можно скачать с FTP-сервера издательства по ссылке **ftp://ftp.bhv.ru/9785977539869.zip** или со страницы книги на сайте **www.bhv.ru**.

Архив имеет следующую структуру:

- папка Listings:
  - файл HTML.doc — все листинги *главы 1 «Основы HTML 5. Создаем дизайн сайта»*;
  - файл CSS.doc — все листинги *главы 2 «Основы CSS 3. Форматируем Web-страницу с помощью стилей»*;
  - файл JavaScript.doc — все листинги *главы 3 «Основы JavaScript. Создаем страницы, реагирующие на действия пользователей»*;
  - файл Setup.doc — все листинги *главы 4 «Программное обеспечение Web-сервера. Устанавливаем и настраиваем программы под Windows»*;
  - файл PHP.doc — все листинги *главы 5 «Основы PHP. Создаем динамические Web-страницы»*;
  - файл MySQL.doc — все листинги *главы 6 «Основы MySQL. Работаем с базами данных»*;
  - файл AJAX.doc — все листинги *главы 7 «AJAX. Обмен данными без перезагрузки Web-страницы»*;
- файл Practice.doc — дополнительная *глава «Сплошная практика. Создаем динамический сайт»*;
- папка Site — содержит все файлы листингов главы *«Сплошная практика. Создаем динамический сайт»*;
- файл Readme.txt — описание электронного архива.

Настоятельно рекомендуем обязательно рассматривать все примеры из книги и вначале самостоятельно набирать код. При наборе вы создадите множество ошибок. Именно умение находить эти ошибки сделает из вас настоящего Web-мастера.

При отсутствии программы MS Word открыть прилагаемые файлы можно с помощью бесплатного текстового редактора Writer пакета LibreOffice (**https://ru.libreoffice.org/**).

Предметный указатель

$

@

_

<

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U



V

W





Z

А

Б

В

Г

Д

Е

З

И

К

С

Т



У



Ф



Х

Ц

Ч

Ш

Э

Я